PA 01-308

reference 1

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（A）　(11)特許出願公開番号

特開平8－16429

(43)公開日　平成8年(1996)1月19日

(51)Int.Cl.⁶	識別記号	庁内整理番号	FI	技術表示箇所
G06F 11/28		A 7313－5B		
9/06	540	S 7230－5B		
9/38	370	X		
15/16	430	A		
	450	Z		

審査請求　未請求　請求項の数50　FD　（全 67 頁）

(21)出願番号　特願平7－127577

(22)出願日　平成7年(1995)4月28日

(31)優先権主張番号　特願平6－114668

(32)優先日　平6(1994)4月28日

(33)優先権主張国　日本（ＪＰ）

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地

(72)発明者　内平　直志
神奈川県川崎市幸区柳町70番地　株式会社東芝柳町工場内

(72)発明者　本位田　真一
神奈川県川崎市幸区柳町70番地　株式会社東芝柳町工場内

(72)発明者　大須賀　昭彦
神奈川県川崎市幸区柳町70番地　株式会社東芝柳町工場内

(74)代理人　弁理士　鈴江　武彦

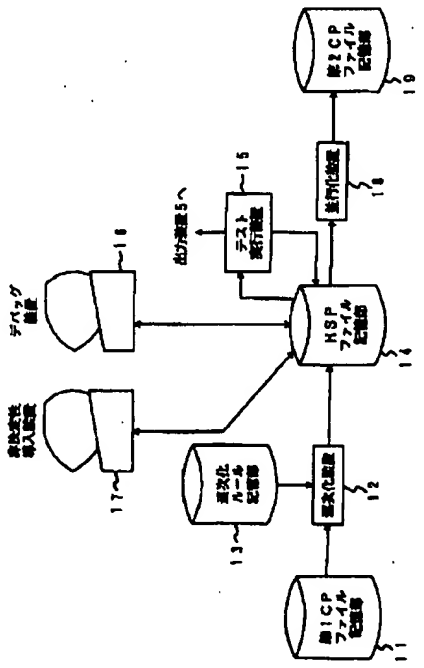
最終頁に続く

(54)【発明の名称】　並行プログラム作成支援装置及び並行プログラム作成方法並びに並行プログラム実行装置

(57)【要約】

【目的】　並行プログラムのテスト・デバッグを容易に実現でき、並行プログラムの開発を効率よく行うことができる並行プログラムの作成方法及びその作成支援装置並びに効果的なテスト・デバッグ及び再現性を保証した部分並行実行を可能とする並行プログラム実行装置を提供すること。

【構成】　本発明の並行プログラム作成支援装置は、並行構造を有する第１並行プログラムを逐次実行可能な逐次プログラムに変換する逐次化手段１２と、前記逐次プログラムのテスト・デバッグを行い、テスト・デバッグ情報を作成するテスト・デバッグ手段１６と、テスト・デバッグ後の前記逐次プログラムを前記テスト・デバッグ情報に基づいて並行化することにより第２並行プログラムに変換する並行化手段１８とを備えた。また、前記テスト・デバッグ手段は、前記逐次プログラムに対して並行性に関する情報を導入する手段を含む。



【特許請求の範囲】

【請求項１】並行構造を有する第１並行プログラムを逐次実行可能な逐次プログラムに変換する逐次化手段と、前記逐次プログラムのテスト・デバッグを行い、テスト・デバッグ情報を作成するテスト・デバッグ手段と、テスト・デバッグ後の前記逐次プログラムを前記テスト・デバッグ情報に基づいて並行化することにより第２並行プログラムに変換する並行化手段と、を具備することを特徴とする並行プログラム作成支援装置。

【請求項２】　前記テスト・デバッグ手段は、前記逐次プログラムに対して並行性に関する情報を導入する手段を含むことを特徴とする請求項１記載の並行プログラム作成支援装置。

【請求項３】　前記逐次プログラムは、セクション情報と、プログラム構造情報と、逐次化情報とを含むことを特徴とする請求項１記載の並行プログラム作成支援装置。

【請求項４】　前記第１並行プログラムの並行構造を解析する解析手段を更に具備し、前記並行化手段は、前記テスト・デバッグ手段によりテスト・デバッグが行われた前記逐次プログラムを前記解析手段の解析結果を用いて並行化して、第２並行プログラムに変換する手段を含むことを特徴とする請求項１記載の並行プログラム作成支援装置。

【請求項５】並行構造を有する第１並行プログラムの所定のセクション群を逐次実行可能な逐次プログラムに変換する逐次化手段と、
前記第１並行プログラムの所定のセクション群の並行構造を解析する並行構造解析手段と、
前記逐次プログラムの所定のセクション群の逐次構造を解析する逐次構造解析手段と、
前記並行構造解析手段で解析された並行構造に関する相互関係及び前記逐次構造解析手段で解析された逐次構造に関する相互関係をグラフ情報として表示するグラフ情報表示手段と、
前記逐次プログラムのうち選択された所定のセクション群を並行化して部分的に逐次構造を有する部分逐次プログラムに変換する部分逐次化手段と、
前記部分逐次プログラムを並行化して第２並行プログラムに変換する並行化手段と、を具備することを特徴とする並行プログラム作成支援装置。

【請求項６】　前記逐次化手段は、
前記第１並行プログラムを実行する実行手段と、
前記実行手段による実行ログを保存する保存手段と、
前記保存手段により保存された実行ログ及び前記第１並行プログラムを解析する解析手段と、
前記保存手段により保存された実行ログを前記解析手段による解析結果に基づいて並べ替えする並べ替え手段と、を含むことを特徴とする請求項１ないし請求項５のいずれかに記載の並行プログラム作成支援装置。

【請求項７】　前記実行手段は、
前記第１並行プログラムを解析してセクション情報の抽出をおこなうプログラム解析手段と、
前記セクション情報を記憶するセクション情報記憶手段と、
前記セクション情報記憶手段に記憶されたセクション情報を変更するセクション情報変更手段と、
前記セクション情報記憶手段により記憶されたセクション情報に基づいて前記並行プログラムの実行を行なう並行プログラム実行手段と、を含むことを特徴とする請求項６記載の並行プログラム作成支援装置。

【請求項８】　前記導入手段は、
前記逐次プログラムのプロセスの流れを制約と遷移条件からなるフィールドに変換して該フィールドを表すフィールドデータを生成するフィールドデータ生成手段と、
前記手段により生成されたフィールドデータで表されるフィールドをチューニングするためのチューニング手段と、
前記フィールドデータ生成手段により生成されたフィールドデータで表されるフィールドを表示する表示手段と、含むことを特徴とする請求項１ないし請求項５のいずれかに記載の並行プログラム作成支援装置。

【請求項９】前記逐次化手段は、前記第１並行プログラムを複数の超逐次プログラムに変換する手段を含み、
前記テスト・デバッグ手段は、前記複数の超逐次プログラムを独立にテスト・デバッグする手段を含むことを特徴とする請求項１ないし請求項４のいずれかに記載の並行プログラム作成支援装置。

【請求項１０】並行構造を有する第１並行プログラムの所定のプロセス群を所定の実行順序に従って逐次実行可能な逐次プログラムに変換する逐次化手段と、
前記逐次構造プログラムに対して並行化の候補となるプロセス群を指定するプロセス群指定手段と、
前記手段により指定されたプロセス群の実行順序を入れ換えて前記逐次プログラムを複数の並行模擬プログラムに変換する模擬並行プログラム変換手段と、
前記複数の並行模擬プログラムを部分的に逐次構造を有する１つの部分逐次プログラムに変換する部分逐次化手段と、
前記部分逐次プログラムを並行化して第２並行プログラムに変換する並行化手段とを具備することを特徴とする並行プログラム作成支援装置。

【請求項１１】逐次化ルールを格納する逐次化ルール格納手段と、
前記逐次化ルール格納手段に格納された逐次化ルールに従って並行構造を有する第１並行プログラムを逐次実行可能な逐次プログラムに変換する逐次化手段と、
前記逐次プログラムに対して並行性に関する情報を導入すべく前記逐次化ルール格納手段に格納された逐次化ルールを修正する逐次化ルール修正手段と、

前記逐次プログラムのテスト・デバッグを行うためのテスト・デバッグ手段と、
前記逐次化ルール修正手段により並行性に関する情報が導入されかつ前記テスト・デバッグ手段によりテスト・デバッグが行われた逐次プログラムを並行化して第２並行プログラムに変換する並行化手段と、を具備することを特徴とする並行プログラム作成支援装置。
【請求項１２】　プロセス群がメッセージ情報を交換しながら並行して動作する実行環境に用いられる並行プログラムの作成を支援する並行プログラム作成支援装置において、
第１並行プログラムのプロセス群の実行履歴であるログ情報を逐次化ルールとして取得するログ情報取得手段と、
前記ログ情報取得手段により取得されたログ情報を記憶するログ情報記憶手段と、
前記ログ情報記憶手段に記憶されているログ情報を修正するためのログ情報修正手段と、
前記ログ情報記憶手段に記憶されているログ情報に基づいて前記プロセス群を逐次的に起動制御するプロセス群制御手段と、
前記ログ情報記憶手段に記憶されたログ情報を並行化して第２並行プログラムに変換する並行化手段と、を具備することを特徴とする並行プログラム作成支援装置。
【請求項１３】　前記ログ情報修正手段は、
前記ログ情報記憶手段に記憶されているログ情報を時系列に表示する表示手段と、
前記表示手段により時系列に表示されたログ情報内のデータの順序の入れ換えを指示するための入れ替え指示手段と、
前記入れ替え指示手段による指示に従って前記ログ情報記憶手段に記憶されているログ情報を書き換える書換手段と、を含むことを特徴とする請求項１２記載の並行プログラム作成支援装置。
【請求項１４】　前記ログ情報修正手段は、プロセス間の処理タイミングの非決定性を導入する非決定性導入手段を含むことを特徴とする請求項１２記載の並行プログラム作成支援装置。
【請求項１５】　プロセス群がメッセージ情報を交換しながら並行して動作できる実行環境で前記プロセス群が実行順序規定情報に従って動作する並行プログラムを作成支援する並行プログラム作成支援装置において、
前記実行順序規定情報を分割する実行順序情報分割手段と、
前記実行順序情報分割手段によって分割された実行順序情報に基づいて前記プロセスを起動制御するプロセス制御手段と、を具備することを特徴とする並行プログラム作成支援装置。
【請求項１６】　前記実行順序規定情報を分割するための基準を与える分割基準指定手段を保持する手段を更に具備することを特徴とする請求項１５記載の並行プログラム作成支援装置。
【請求項１７】　前記プロセス制御手段は、前記メッセージ交換の実行履歴であるログ情報を実行順序規定情報として用いるとともに、前記分割基準指定手段における分割基準として前記メッセージ中の宛先プロセス情報を基準として用いることを特徴とする請求項１５記載の並行プログラム作成支援装置。
【請求項１８】　前記プロセス制御手段を各プロセス毎に保持する手段を具備することを特徴とする請求項１７記載の並行プログラム作成支援装置。
【請求項１９】　前記実行順序情報分割手段によって分割された実行順序情報をそれぞれ別々に保存する分割実行順序情報保存手段を更に具備し前記プロセス制御手段は該プロセスに対応する分割実行順序情報保存手段に保存されている分割実行順序情報に基づいてプロセスを起動制御する手段を含むことを特徴とする請求項１８記載の並行プログラム作成支援装置。
【請求項２０】　前記プロセス制御手段は、前記プロセス群の実行履歴であるログ情報を前記実行順序規定情報として用いることを特徴とする請求項１５記載の並行プログラム作成支援装置。
【請求項２１】並行構造を有する第１並行プログラムを逐次的にテスト実行する逐次的テスト実行手段と、
前記テスト実行手段によるテスト実行の結果、正常に実行終了したテスト実行結果のバグのない逐次的実行ログを蓄積する実行ログ蓄積手段と、
前記実行ログ蓄積手段により蓄積された逐次的実行ログを並行化して第２並行プログラムに変換する並行化手段と、を具備することを特徴とする並行プログラム作成支援装置。
【請求項２２】並行構造を有する第１並行プログラムをテスト実行するテスト実行手段と、
前記テスト実行手段によるテスト実行の結果バグのない実行ログを蓄積する第１の実行ログ蓄積手段と、
前記テスト実行手段によるテスト実行の結果バグのある実行ログを蓄積する第２の実行ログ蓄積手段と、
前記第２の実行ログ蓄積手段により蓄積された実行ログを修正し、修正された実行ログを前記第１の実行ログ蓄積手段に蓄積する修正手段と、
前記第１の実行ログ蓄積手段により蓄積された実行ログを並行化して第２並行プログラムに変換する並行化手段と、を具備することを特徴とする並行プログラム作成支援装置。
【請求項２３】並行構造を有する第１並行プログラムを逐次実行可能な逐次プログラムに変換する第１ステップと、
前記逐次プログラムのテスト・デバッグを行い、テスト・デバッグ情報を作成する第２ステップと、
テスト・デバッグ後の前記逐次プログラムを前記テスト

・デバッグ情報に基づいて並行化することにより第２並行プログラムに変換する第３ステップと、を具備することを特徴とする並行プログラム作成方法。

【請求項２４】　前記第２ステップは、前記逐次プログラムに対して並行性に関する情報を導入するサブステップを含むことを特徴とする請求項２３記載の並行プログラム作成方法。

【請求項２５】前記第１ステップは、前記第１並行プログラムの並行構造を解析するサブステップを含み、
前記第３ステップは、テスト・デバッグ後の前記逐次プログラムを前記第１並行プログラムの並行構造を用いて並行化するサブステップとを含むことを特徴とする請求項２３記載の並行プログラム作成方法。

【請求項２６】　前記第２ステップは、前記逐次プログラムに対して指定された非決定性を前記並行性に関する情報として導入するサブステップを含むことを特徴とする請求項２３ないし請求項２５のいずれかに記載の並行プログラム作成方法。

【請求項２７】前記第１ステップは、
並行構造を有する前記第１並行プログラムの所定のセクション群を逐次実行可能な逐次プログラムに変換する第１サブステップと、
前記第１並行プログラムの所定のセクション群の並行構造を解析する第２サブステップと、
前記逐次プログラムの所定のセクション群の逐次構造を解析する第３サブステップと、
前記第２サブステップで解析された並行構造に関する相互関係及び前記第３サブステップで解析された逐次構造に関する相互関係をグラフ情報として表示する第４サブステップと、
前記逐次プログラムのうち選択された所定のセクション群を並行化して部分的に逐次構造を有する部分逐次プログラムに変換する第５サブステップと、を含み、
前記第３ステップは、前記部分逐次プログラムを並行化して第２並行プログラムに変換するサブステップを含むことを特徴とする請求項２３記載の並行プログラム作成方法。

【請求項２８】並行構造を有する前記第１並行プログラムの所定のセクション群を逐次実行可能な逐次プログラムに変換する第１ステップと、
前記第１並行プログラムの所定のセクション群の並行構造を解析する第２ステップと、
前記逐次プログラムの所定のセクション群の逐次構造を解析する第３ステップと、
前記第２ステップで解析された並行構造に関する相互関係及び前記第３ステップで解析された逐次構造に関する相互関係をグラフ情報として表示する第４ステップと、
前記逐次プログラムのうち選択された所定のセクション群を並行化して部分的に逐次構造を有する部分逐次プログラムに変換する第５ステップと、
前記部分逐次プログラムを並行化して第２並行プログラムに変換する第６ステップと、を具備することを特徴とする並行プログラム作成方法。

【請求項２９】前記第１サブステップは、前記逐次プログラムの所望の実行結果が得られるまで当該逐次プログラムのテスト・デバッグを行うサブステップを含むことを特徴とする請求項２８記載の並行プログラム作成方法。

【請求項３０】　前記第４サブステップは、前記所定のセクション群をノードとし、前記並行構造に関する相互関係を第１アークとし、前記逐次構造に関する相互関係を第２アークとして前記グラフ情報を表示するサブステップを含むことを特徴とする請求項２８記載の並行プログラム作成方法。

【請求項３１】　前記第５サブステップは、前記部分逐次プログラムの所望の実行結果が得られるまで該部分逐次プログラムのテスト・デバッグを行うサブステップを含むことを特徴とする請求項２８記載の並行プログラム作成方法。

【請求項３２】前記第５サブステップは、前記部分逐次プログラムにおける所定のセクション群の逐次構造を解析するサブステップを含み、
前記第１ステップは、前記第４サブステップから第５サブステップを所定の回数繰り返すサブステップを更に含むことを特徴とする請求項２８記載の並行プログラム作成方法。

【請求項３３】前記第１ステップは、
前記第１並行プログラムを実行する第１サブステップと、
前記第１サブステップによる実行ログを保存する第２サブステップと、
前記第２サブステップにより保存された実行ログ及び前記第１並行プログラムを解析する第３サブステップと、
前記第２サブステップにより保存された実行ログを前記第３サブステップによる解析結果に基づいて並べ替える第４サブステップと、を含むことを特徴とする請求項２３ないし請求項２５、請求項２８又は請求項２９のいずれかに記載の並行プログラム作成方法。

【請求項３４】　前記第３サブステップは、前記第２サブステップにより保存された実行ログ及び前記第１並行プログラムからプロセスの先行関係を抽出し先行関係情報として保存するサブステップを含むことを特徴とする請求項３３記載の並行プログラム作成方法。

【請求項３５】　前記第２ステップは、前記逐次プログラムのプロセスの流れを制約と遷移条件からなるフィールドに変換するサブステップと、前記フィールドをチューニングするサブステップと、前記並行性に関する情報を導入するサブステップとを含むことを特徴とする請求項２３ないし請求項２５のいずれかに記載の並行プログラム作成方法。

【請求項３６】複数のプロセス群を有し並行構造を有する第１並行プログラムの所定のプロセス群を所定の実行順序に従って逐次実行可能な逐次プログラムに変換する第１ステップと、
前記逐次プログラムに対して並行化の候補となるプロセス群を指定する第２ステップと、
前記第２ステップにより指定されたプロセス群の実行順序を入れ換えて前記逐次プログラムを複数の並行模擬プログラムに変換する第３ステップと、
前記複数の並行模擬プログラムを部分的に逐次構造を有する１つの部分逐次プログラムに変換する第４ステップと、
前記部分逐次プログラムを並行化して第２並行プログラムに変換する第５ステップと、を具備することを特徴とする並行プログラム作成方法。
【請求項３７】　前記第１ステップは、前記逐次プログラムの所望の実行結果が得られるまで該逐次プログラムのテスト・デバッグを行うサブステップを含むことを特徴とする請求項３６記載の並行プログラム作成方法。
【請求項３８】　前記第２ステップは、
前記第１並行プログラムを解析するサブステップと、
前記解析結果から前記並行化の候補となるプロセス群を抽出するサブステップと、を含むことを特徴とする請求項３６記載の並行プログラム作成方法。
【請求項３９】　前記第３ステップは、前記複数の並行模擬プログラムの所望の実行結果が得られるまで前記複数の並行模擬プログラムのテスト・デバッグを行うサブステップを含むことを特徴とする請求項３６記載の並行プログラム作成方法。
【請求項４０】　前記第３ステップは、前記複数の並行模擬プログラムの一部の実行結果から不要と判断される並行模擬プログラムを取り除くサブステップを含むことを特徴とする請求項３６記載の並行プログラム作成方法。
【請求項４１】　前記第４ステップは、前記部分逐次プログラムに対して並行化の候補となるプロセス群を指定するサブステップを含み、
前記第３ステップから前記第４ステップを所定の回数繰り返すステップを更に具備することを特徴とする請求項３６記載の並行プログラム作成方法。
【請求項４２】　前記第１ステップは、
前記第１並行プログラムを逐次化ルールに従って前記逐次プログラムに変換する第１サブステップと、
前記逐次プログラムに対して並行性に関する情報を導入するために前記逐次化ルールを修正する第２サブステップと、を含み、
前記第３ステップは、前記テスト・デバッグ情報と、前記並行性に関する情報とに基づいて前記逐次プログラムを並行化して第２並行プログラムに変換するサブステップを含むことを特徴とする請求項２３記載の並行プログラム作成方法。
【請求項４３】並行構造を有する第１並行プログラムをテスト実行する第１ステップと、
前記第１ステップによるテスト実行の結果バグのない実行ログを蓄積する第２ステップと、
前記第２ステップにより蓄積された前記実行ログを並行化して第２並行プログラムに変換する第３ステップと、を具備することを特徴とする並行プログラム作成方法。
【請求項４４】前記第１ステップによるテスト実行の結果バグのある実行ログを蓄積する第４ステップと、
前記第４ステップにより蓄積された実行ログに基づいて前記第１並行プログラムを修正して、再度実行ログを蓄積する第５ステップと、を更に具備することを特徴とする請求項４３記載の並行プログラム作成方法。
【請求項４５】並行プログラムを解析してセクション情報の抽出をおこなうプログラム解析手段と、
前記セクション情報を記憶するセクション情報記憶手段と、
前記セクション情報記憶手段により記憶されたセクション情報に基づいて前記並行プログラムの実行を行なう並行プログラム実行手段と、を具備することを特徴とする並行プログラム実行装置。
【請求項４６】　前記セクション情報記憶手段によって記憶されるセクション情報を編集するセクション情報編集手段及び前記セクション情報記憶手段によって記憶されるセクション情報内の各セクションをオブジェクトに変換するコンパイル手段の少なくとも一方を更に具備することを特徴とする請求項４５記載の並行プログラム実行装置。
【請求項４７】　前記プログラム解析手段が、入力されたプログラムを中間言語に変換する手段と、前記中間言語からセクションを抽出する手段とを更に含むことを特徴とする請求項４５記載の並行プログラム実行装置。
【請求項４８】　前記実行手段が、実行するセクションの候補を記憶する実行セクション候補記憶手段と、前記実行セクション候補記憶手段によって記憶されるセクションから次に実行するセクションを選択する実行セクション選択手段と、前記実行セクション選択手段によって選択されたセクションを実行するセクション実行手段とを更に含むことを特徴とする請求項４５記載の並行プログラム実行装置。
【請求項４９】　前記実行セクション選択手段が、実行可能なセクションを一つ選択するモードと実行可能なセクションをすべて選択するモードを含むことを特徴とする請求項４８記載の並行プログラム実行装置。
【請求項５０】　前記実行セクション実行手段が、前記実行セクション選択手段によって選択されたセクションが複数の時はそれらのセクションを並行に実行することを特徴とする請求項４８記載の並行プログラム実行装置。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、並行プログラムの作成方法及びその作成支援装置並びに並行プログラム実行装置に関する。

【０００２】

【従来の技術】近年の半導体集積回路技術の進歩により、複雑なプロセッサ及び大容量のメモリが小型かつ低価格で実現できるようになり、多数のプロセッサからなる並行処理システムや分散処理システムが実用化されている。このようなハードウェアに対しては、専用のプログラム、即ち並列プログラムや分散処理プログラム等（以下「並行プログラム」という）を用いなければならない。従って、並行プログラムをいかに効率よく開発するかは、優れたアルゴリズムを検討する場合と同様に重要な課題となっている。

【０００３】ところで、プログラム開発においては、プログラム中のバグを見つけ修正すること（即ちテスト・デバッグ）と呼ばれる開発工程がプログラム開発の効率に大きく影響する。しかし、並行プログラムの開発においては、逐次プログラムの開発においては遭遇することのない、並行プログラム特有の問題を考慮する必要がある。この並行プログラム特有の問題とは、並行プログラムを構成する各プロセスは、各プロセス間における相互作用のタイミングにより様々な振る舞いをする可能性があるため、並行プログラム全体が正しく動作しないという問題である。この問題は、並行プログラムの性質に基づく問題であり、一般に「非決定性」と呼ばれる。

【０００４】例えば、図１１６に示す並行プログラムを考慮する。図１１６（ａ）において、プロセスＰ１は、共有メモリＭの初期設定（ｉｎｉｔ）を行うプロセス、プロセスＰ２は、共有メモリＭから読み出し（ｒｅａｄ）を行うプロセス、プロセスＰ３は、共有メモリＭに書き込み（ｗｒｉｔｅ）を行うプロセスを示す。これらのプロセスをそれぞれ異なるプロセッサで実行するような並列処理システム等で動作させた場合、全部で６通り（図１１６（ｂ）参照）の動作の組み合わせがあることになる。通常、システムは初期設定で処理を開始するから、今、プロセスＰ１（ｉｎｉｔ）→Ｐ２（ｒｅａｄ）→Ｐ３（ｗｒｉｔｅ）又はＰ１（ｉｎｉｔ）→Ｐ３（ｗｒｉｔｅ）→Ｐ２（ｒｅａｄ）の順番でプログラムが動作する場合に正しい結果が得られるものとすれば、残りの４通り（例えばＰ２→Ｐ３→Ｐ１）は、初期化が最初に行われないため、明らかに正しい結果が得られないことがわかる。

【０００５】上記のようにプロセスの振る舞いに関する非決定性は、並行プログラムを動作させる毎に、その時点におけるシステムの状況等によって結果を異なったものになる。従って、この非決定性に関する問題を解決しない限り、その並行プログラムは、テストにおいてたとえ正常に動作することがあっても、常に正常に動作するという保証はない。

【０００６】また、非決定性に関するバグは、一般に、逐次プログラムにおけるバグを見つける場合よりも困難である。なぜなら、逐次プログラムにおいては、テスト・デバッグ時にプログラム中の全てのパスを実行することによって動作を確認することができるのに対し、並行プログラムにおいては、全ての組合せ的なパス（即ち、各プロセス中の全てのパスのみならず、プロセス相互間の振る舞い）を考慮してパスを実行しなければならないからである。上記の例のようにプロセスの数が少ない場合にあっては、各プロセス相互間の振る舞いを全て列挙することは比較的容易であるが、実際のプログラム開発では、その数は膨大になり、その組み合わせも膨大なものとなるため、全ての振る舞いを把握することはもはや不可能なものとなる。

【０００７】

【発明が解決しようとする課題】上記のように、並行プログラム開発におけるテスト・デバッグは、逐次プログラム開発におけるテスト・デバッグに比較して、非常に困難である。特に、プログラム自体が巨大化した今日においては、このテスト・デバッグは一層困難になっている。

【０００８】更に、並行プログラム特有の問題である非決定性によって再現性が保証されないことからテスト・デバッグが困難となり、また予期せぬエラーが出現するという問題を有する。

【０００９】本発明は、並行プログラムのテスト・デバッグを容易に実現でき、並行プログラムの開発を効率よく行うことができる並行プログラムの作成方法及びその作成支援装置を提供することを目的とする。

【００１０】本発明の他の目的は、並行プログラムを逐次的に実行することにより再現性を保証し、効果的なテスト・デバッグを行なうことを可能とすると共に、テスト・デバッグ後に一部分を並行化して実行することにより、非決定性による予期せぬエラーを排除した安全な並行プログラムの実行が可能な並行プログラム実行装置を提供することである。

【００１１】本発明は並行プログラムを一旦逐次化し、逐次化したプログラムに対してテスト・デバッグを行い、テスト・デバッグが完了した時点で、プログラムの並行性を復元することを骨子とする。

【００１２】

【課題を解決するための手段】本発明は、上記の課題を解決するために次のような手段を講じた。並行プログラムのプログラミングの困難さは、「人間の思考は本来逐次的であり、並行に動くものをありのままでは論理的に認識することは困難である」ことに起因する。そこで、本発明は並行プログラムをいったん逐次化し、逐次化したプログラムに対してプログラミング、テスト・デバッ

グを行う。これは、従来の逐次プログラミングと同じレベルの困難さである。そして、テスト・デバッグが完了した時点で並行性を、そのテスト・デバッグ情報を用いて、自動的に復元する。

【００１３】上記のようなプログラミングのスタイルを「超逐次プログラミング」と呼ぶ。この「超逐次プログラミング」によれば、従来の手法におけるプログラミングの困難さを解決できる。本発明の基本コンセプトは、以下の３つのステップ（或いは手段）から構成される。

【００１４】（１）　並行プログラムを逐次化して超逐次プログラムを生成するステップ（手段）。

【００１５】（２）　超逐次プログラムに対して、作業（プログラミング、テスト・デバッグ、並行性の導入）を行うステップ（手段）。

【００１６】（３）　作業が完了した超逐次プログラムを並行化して並行プログラムを作成するステップ（手段）。

【００１７】ここで、「超逐次プログラム」とは、オリジナルの並行プログラムの並行構造に関する情報を保ちながら逐次化したプログラムをいう。

【００１８】本発明に係る並行プログラム作成装置は、並行構造を有する第１並行プログラムを逐次実行可能な逐次プログラムに変換する逐次化手段と、前記逐次プログラムのテスト・デバッグを行い、テスト・デバッグ情報を作成するテスト・デバッグ手段と、テスト・デバッグ後の前記逐次プログラムを前記テスト・デバッグ情報に基づいて並行化することにより第２並行プログラムに変換する並行化手段とを具備することを特徴とする。また、本発明に係る並行プログラムの作成方法は、並行構造を有する第１並行プログラムを逐次実行可能な逐次プログラムに変換する第１ステップと、前記逐次プログラムのテスト・デバッグを行い、テスト・デバッグ情報を作成する第２ステップと、テスト・デバッグ後の前記逐次プログラムを前記テスト・デバッグ情報に基づいて並行化することにより第２並行プログラムに変換する第３ステップとを具備することを特徴とする。

【００１９】本発明の好ましい態様を以下に列挙する。

（１）　前記逐次プログラムに対して並行性に関する情報を導入すること。この並行性に関する情報は、例えば、後述する良い非決定性に関する情報を含む。

（２）　第１並行プログラムの並行構造を解析し、この並行構造と逐次プログラムをテストして得られた実行結果を用いて逐次プログラムの並行化を行うこと。

【００２０】（３）　第１並行プログラムの逐次プログラムに変換されるセクションの並行構造と、逐次プログラムのセクションの逐次構造をそれぞれ解析し、第１並行プログラムの並行構造に関する相互関係及び逐次プログラムの逐次構造に関する相互関係をグラフ情報として表示する。このグラフ情報の表示においては、所定のセクション群をノードとし、第１並行プログラムの並行構造に関する相互関係を第１アークとし、逐次プログラムの逐次構造に関する相互関係を第２アークとしてグラフ情報を表示する。そして、逐次プログラムのうち選択されたセクションを並行化して部分的に逐次構造を有する部分逐次プログラムに変換し、この部分逐次プログラムを並行化して第２並行プログラムに変換する。

【００２１】ここで、第１並行プログラムを逐次プログラムに変換するステップにおいて、逐次プログラムの所望の実行結果が得られるまで逐次プログラムのテスト・デバッグを行うステップを含ませるか、又は逐次プログラムを部分逐次プログラムに変換するステップにおいて、所望の実行結果が得られるまで部分逐次プログラムのテスト・デバッグを行うステップを含ませるか、或いはこれら両方のテスト・デバッグステップを含ませてもよい。

【００２２】更に、逐次プログラムを部分逐次プログラムに変換するステップに、部分逐次プログラムの逐次構造を解析するステップを含ませた上で、グラフ情報の表示と並行化セクションの選択及び部分逐次プログラムへの変換のステップを所定回数繰り返すようにしてもよい。

【００２３】（４）　第１並行プログラムを逐次プログラムに変換する逐次化に際して、この第１並行プログラムを実行してそれによる実行ログを保存するとともに、この保存された実行ログ及び第１並行プログラムを解析し、この解析結果に基づいて、保存された実行ログを並べ替える。実行ログ及び第１並行プログラムの解析においては、例えば保存された実行ログ及び第１並行プログラムからプロセスの先行関係を抽出し、これを先行関係情報として保存する。

【００２４】（５）　逐次プログラムに対する並行性に関する情報の導入に際して、逐次プログラムのプロセスの流れを制約と遷移条件からなるフィールドに変換し、このフィールドをチューニングすることによって行う。更に、このフィールドを表すフィールドデータを表示する。

【００２５】（６）　逐次プログラムに対して並行化の候補となるプロセス群を指定し、このプロセス群の実行順序を入れ換えて逐次プログラムを複数の並行模擬プログラムに変換した後、これら複数の並行模擬プログラムを部分的に逐次構造を有する１つの部分逐次プログラムに変換し、この部分逐次プログラムを並行化して第２並行プログラムに変換する。

【００２６】ここで、第１並行プログラムを逐次プログラムに変換するステップにおいて、逐次プログラムの所望の実行結果が得られるまで該逐次プログラムのテスト・デバッグを行うステップを含ませるか、又は複数の並行模擬プログラム群を１つの部分逐次プログラムに変換するステップにおいて、並行模擬プログラム群の所望の実行結果が得られるまで複数の並行模擬プログラムのテ

スト・デバッグを行うか、或いはこれら両方のテスト・デバッグステップを含ませてもよい。また、逐次プログラムに対して並行化の候補となるプロセス群を指定する際、第１並行プログラムを解析し、この解析結果から並行化の候補となるプロセス群を抽出してもよい。また、逐次プログラムを複数の並行模擬プログラムに変換するステップにおいて、複数の並行模擬プログラムの一部の実行結果から不要と判断される並行模擬プログラムを取り除くステップを含ませてもよい。更に、複数の並行模擬プログラムを１つの部分逐次プログラムに変換するステップにおいて、部分逐次プログラムに対して並行化の候補となるプロセス群を指定するステップを含ませ、逐次プログラムを複数の並行模擬プログラムに変換するステップと、複数の並行模擬プログラムを１つの部分逐次プログラムに変換するステップを所定の回数繰り返してもよい。

【００２７】（７）　第１並行プログラムの逐次プログラムへの変換を所定の逐次化ルールに従って行う場合、この逐次化ルールを修正することによって、逐次プログラムに対して並行性に関する情報を導入する。

【００２８】（８）　プロセス群がメッセージ情報を交換しながら並行して動作する実行環境に用いられる並行プログラムの作成を支援する並行プログラム作成支援装置において、並行プログラムのプロセス群の実行履歴であるログ情報を逐次化ルールとして取得して記憶し、この記憶したログ情報を修正可能とする。記憶されているログ情報に基づいてプロセス群を逐次的に起動制御し、また記憶されたログ情報を並行化して第２並行プログラムに変換する。

【００２９】ここで、ログ情報修正手段は、前記ログ情報記憶手段に記憶されているログ情報を時系列に表示する表示手段と、前記表示手段により時系列に表示されたログ情報内のデータの順序の入れ換えを指示するための入れ替え指示手段と、前記入れ替え指示手段による指示に従って前記ログ情報記憶手段に記憶されているログ情報を書き換える書換手段とを含むことを特徴とする。また、ログ情報修正手段は、プロセス間の処理タイミングの非決定性を導入する非決定性導入手段を含むことを特徴とする。

【００３０】（９）　プロセス群がメッセージ情報を交換しながら並行して動作できる実行環境で前記プロセス群が実行順序規定情報に従って動作するシステムにおいて、実行順序規定情報を分割し、この分割された実行順序情報に基づいてプロセス群を起動制御する。

【００３１】この場合、実行順序規定情報を分割するための基準を与える分割基準指定手段を保持する手段を更に具備してもよい。また、メッセージ交換の履歴を実行順序規定情報として用いるとともに、分割基準指定手段における分割基準としてメッセージ中の宛先プロセス情報を基準として用いる。また、メッセージ交換の履歴を実行順序規定情報として用い、分割基準指定手段における分割基準として前記メッセージ中の宛先プロセス情報を基準として用いるとともに、プロセス制御手段を各プロセス毎に保持する手段を具備する。更に、メッセージ交換の履歴を実行順序規定情報として用い、分割基準指定手段における分割基準として前記メッセージ中の宛先プロセス情報を基準として用いるとともに、プロセス制御手段を各プロセス毎に保持する手段と、実行順序情報分割手段によって分割された実行順序情報をそれぞれ別々に保存する分割実行順序情報保存手段を具備し、プロセス制御手段は該プロセスに対応する分割実行順序情報保存手段に保存されている分割実行順序情報に基づいてプロセスを起動制御する。また、プロセス群の実行履歴情報であるログ情報を実行順序規定情報として用いてもよい。

【００３２】（１０）　第１並行プログラムをテスト実行して、そのテスト実行の結果バグのない実行ログを蓄積し、この蓄積されたバグのない実行ログのみを並行化して第２並行プログラムに変換する。更に、テスト実行の結果バグのある実行ログを蓄積し、この蓄積されたバグのある実行ログに基づいて第１並行プログラムを修正する。

【００３３】また、本発明の並行プログラム実行装置は、並行プログラムを解析してセクションを抽出し、それらのセクション間の実行時の同期を制御するための実行順序に関するルールを抽出するプログラム解析手段と、実行時のセクション間の同期を調節するために実行順序に関するルールを編集し、主に実行の効率化のために、抽出された当該セクションを融合・分割する編集手段と、セクションをオブジェクトに変換するコンパイル手段と、該実行順序に関するルールに従いセクション単位で逐次又は並行に実行を行なう実行手段を具備する。

【００３４】

【作用】上記手段を講じた結果、次のような作用が生じる。本発明では並行プログラムを一旦逐次化し、逐次化したプログラムに対してテスト・デバッグを行うことにより、従来の並行プログラムのプログラミングより遥かに容易な逐次プログラミングと同じレベルの困難さで並行プログラムのテスト・デバッグが可能となる。

【００３５】また、本発明では、いったん逐次化されたプログラムに対して意図的に並行性に関する情報（良い非決定性）を導入できるので、意図しない並行性（悪い非決定性）に基づいて発生するバグを回避でき、高信頼化が達成できる。

【００３６】上記のグラフ情報によって並行化及び逐次化の情報をユーザに対して同時に提示することにより、ユーザは第１並行プログラムの並行構造を考慮しつつ、良い非決定性部分の指定をすることができるようになる。また、並行プログラム記述レベルにおける良い非決定性部分の指定・解除ではなく、グラフ情報に対して良

い非決定性部分を指定・解除を行うことでができるため、高度な並行プログラミング技術を必要とすることなく、容易に並行プログラムの開発をすることができるようになる。

【００３７】第１並行プログラムを逐次プログラムに変換する際、第１並行プログラム及びその実行ログを解析し、この解析結果に基づいて実行ログを並べ替えるようにすれば、並べ替え後の実行ログを表示してユーザに提示することによって、並行プログラムの実行過程の理解が容易となり、テスト・デバッグの効率が向上する。逐次プログラムに並行性に関する情報を導入する際、逐次プログラムのプロセスの流れを制約と遷移条件からなるフィールドに変換し、更にそのフィールドデータを表示することによって、フィールドを対話的・視覚的に編集することで並行性に関する情報を効果的に導入し、バグのない並行プログラムが効率的に作成される。

【００３８】第１並行プログラムから逐次化された逐次プログラムに対して並行化の候補となるプロセス群を指定し、このプロセス群の実行順序を入れ換えて逐次プログラムを複数の並行模擬プログラムに変換した後、これら複数の並行模擬プログラムを部分的に逐次構造を有する１つの部分逐次プログラムに変換し、この部分逐次プログラムを並行化して第２並行プログラムに変換することにより、部分逐次プログラム上で並行プログラムの動作を十分に確認することができる。また、部分逐次プログラムに対して並行化の候補となるプロセス群を指定することで、逐次構造プログラムを段階的に並行プログラムへ変換することができる。更に、並行模擬動作系列に基づく逐次実行で正しく動作することが確認された非決定性のみを許容する並行性に関する情報を導入することで、正しく動作する並行プログラムを得ることができる。これらによって、並行プログラムのテスト・デバッグが容易となる。

【００３９】プロセス群がメッセージ情報を交換しながら並行して動作する実行環境に用いられる並行プログラムの作成を支援する並行プログラム作成支援装置において、第１並行プログラムのプロセス群の実行履歴であるログ情報を逐次化ルールとして取得して記憶し、このログ情報を修正可能とするとともに、記憶されているログ情報に基づいてこれに基づいてプロセス群を逐次的に起動制御し、記憶されたログ情報を並行化して第２並行プログラムに変換することにより、ソースプログラムとしての並行プログラムを修正することなく、ログ情報の修正で処理タイミングの非決定性に起因する不具合を解決できる。これにより、処理タイミングの非決定性が内在する並行／並列／分散プログラムの開発が容易となる。また、ユーザの意図する良い非決定性のみを容易に導入することができるため、並行プログラムとしての柔軟性、再利用性及び拡張性を維持することもできる。

【００４０】プロセス群がメッセージ情報を交換しながら並行して動作できる実行環境でプロセス群が実行順序規定情報に従って動作するシステムにおいて、実行順序規定情報を分割し、つまり並行プログラムを逐次化して得られた全プロセスの集中ログ情報を各プロセス毎に分割し、この分割された実行順序情報に基づいてプロセス群を起動制御することにより、無害の非決定性を自然に導入し、集中ログ情報に基づく逐次プログラムの実行時と同一結果を高い処理効率で得ることが可能となる。

【００４１】第１並行プログラムをテスト実行してそのテスト実行の結果、その１つであるバグのない実行ログを蓄積し、この蓄積されたバグのない実行ログのみを並行化して第２並行プログラムに変換することにより、テストで通過したタイミングだけを許容するようにプログラムが動くようになるため、テストしなかったことで残存したバグに陥ることを回避でき、信頼性が向上する。

【００４２】本発明の並行プログラム実行装置によれば並行プログラムの逐次又は部分並行実行を行なうことが可能となるので、従来の逐次プログラムと同様に再現性を保証することができ、効果的かつ安定したテスト・デバッグを行なうことができる。また、逐次実行から部分並行実行に容易に切替えることができるため、逐次化してテスト・デバッグをおこなったプログラムを非決定性に影響されない安全な並行実行を可能とする。他に、セクションの実行順序に関するルールを編集して実行時のセクション間の同期を制御することにより、効果的なテスト・デバッグを行なえるとともにプログラムの効率化の指針とすることもできる。

【００４３】更に、本発明の並行プログラム実行装置は、本発明の並行プログラム作成支援装置或いは作成方法と組み合わせて用いることもできる。この場合には、並行プログラム実行装置は、第１並行プログラム（ソースプログラム）の実行手段として、機能する。更に、並行プログラム実行装置は、本発明の並行プログラム作成支援装置或いは作成方法により作成された並行プログラムの検証用としても利用できる。

【００４４】

【実施例】以下、図面を参照して本発明の実施例を説明する。本発明の実施例の説明を行う前に、本発明において使用する用語を以下のように定義する。

【００４５】並行システム：論理的な並行性を持つシステム。並行プログラムが動くシステムは並行システムである。並列計算機や分散処理システムは並行システムである。

【００４６】並行プログラム（ＣＰ）：論理的に並行に動くモデルに基づいて記述されたプログラム。複数のＣＰＵから構成される並列計算機上で、論理的にも物理的にも並行に動くプログラム（並列プログラム）は、並行プログラムに含まれる。また、単一のＣＰＵ上で物理的には逐次的に動く場合でも、マルチタスクシステムのように論理的に並行であれば、並行プログラムに含まれ

る。

【００４７】逐次プログラム：論理的に逐次に動くモデルに基づいて記述されたプログラム。

【００４８】超逐次プログラム（ＨＳＰ）：並行プログラムにメタレベル（実行管理レベル）の制御を付加することによって、逐次的に動くようにしたプログラム。例えば、並行プログラムとして記述されたプログラムを、実行管理レベルのスケジューラを付加することによって逐次的に動くようにしたプログラムは、超逐次プログラムである。超逐次プログラムは後述する非決定性を持たないので、外部からの入力が同じならばその挙動は必ず再現性を持つ。また、超逐次プログラムにおいて、部分的に並行性（非決定性）を導入することもできる。部分的に並行性（非決定性）が導入された超逐次プログラムを、特に、部分超逐次プログラム（ＰＨＳＰ）、並行性（非決定性）が全くなく完全に逐次化されたプログラムを完全超逐次プログラムと呼ぶこともある。

【００４９】非決定性：入力が同じにもかかわらず、処理のタイミングによってシステムの挙動が変わること。並行プログラムの実行において、非決定性は本質的側面である。ある意味で、非決定性のないプログラムは論理的に逐次プログラムと等価である。

【００５０】良い非決定性：ユーザの意図する非決定性をいう、即ち、ユーザの仕様、その他実現要求に含まれる非決定性。この非決定性により、外部（環境）からの非決定的な入力に適切に応答できるようになる。

【００５１】悪い非決定性：ユーザが予想しなかった非決定性。ユーザの思考回路は逐次的であるため、設計時には意図しなかったケースが実際の並行プログラムの実行では多々発生する。

【００５２】無害な非決定性：その非決定的な選択肢の選択が、最終的な結果に影響しないもの。「超逐次プログラミング」の並行化装置において、並行化する対象は、この「無害な非決定性」を含む。

【００５３】デフォルト逐次化：実行管理レベルで逐次化制御を行なうこと。

【００５４】並行模擬プログラム：超逐次プログラムにおいて、並行化の候補となる特定の範囲について生成された並行模擬動作系列の集合。

【００５５】・並行模擬動作系列：超逐次プログラムにおいて並行化の候補となる特定の範囲について並行プログラムの動作を逐次プログラム上で模擬することを目的として、逐次実行の順序を任意又は意図的に入れ替えて生成される１つの動作列のこと。一般に、１つの並行化範囲には複数の並行模擬動作系列が生成されるので、その全体を１つの並行模擬プログラムという。超逐次プログラム全体では複数の並行模擬プログラムが生成される。

【００５６】図１は、本発明に係る並行プログラム作成支援装置を実現するためのコンピュータシステムの構成例を示す図である。図１において、Ｎ台のプロセッサ１－１、１－２、…、１－Ｎは、並行プログラムを同時に実行することができ、Ｉ／Ｏインタフェース２を介して共有メモリ３及び周辺装置とアクセスすることができる。周辺装置は、入力装置４と出力装置５及び外部記憶装置６から構成される。入力装置４は、キーボードとポインティングデバイス等とからなり、各種コマンドやデータの入力をするために用いられる。出力装置５は、ＣＲＴディスプレイ等からなり、ソースプログラムやテスト・デバッグ状況に関する情報等をテキスト又はグラフィック表示することにより、ユーザに提示する。ユーザは、これら入力装置４及び出力装置５を用いて、対話的にコンピュータを操作することができる。

【００５７】外部記憶装置６は、磁気ディスクや光磁気ディスク等からなり、ソースプログラムやテスト・デバッグ状況に関する情報を書き込み又は読み出すことができるようになっている。

【００５８】なお、上記説明したコンピュータシステムの構成は、これにこだわる必要はなく、例えば、複数の計算機をネットワークを用いて接続した、いわゆる分散ネットワークを用いて構成してもよい。

【００５９】上記のように構成されたコンピュータシステムにおいて、本発明における並行プログラムの作成は、以下のようにして実現される。

【００６０】（実施例１）実施例１では、テスト・デバッグは部分超逐次プログラムに対して行い、好ましい実施例として良い非決定性の導入を超逐次プログラムに対して行う。なお、以下の実施例において、良い非決定性を導入した例を説明するが、必ずしも良い非決定性を導入しなくても良い。

【００６１】図２は、実施例１に係る並行プログラム作成支援装置の概略構成を示す図である。実施例１に係る並行プログラム作成支援装置は、逐次化装置１２と、テスト実行装置１５と、デバッグ装置１６と、非決定性導入装置１７と、並行化装置１８とを具備する。

【００６２】逐次化装置１２は、第１ＣＰファイル記憶部１１に記憶されたソースプログラム（以下、「第１並行プログラム」と称する）を逐次化ルール記憶部１３に記憶された逐次化ルールに基づいて超逐次プログラムに変換して、その結果がＨＳＰファイル記憶部１４に記憶される。第１ＣＰファイル記憶部１１には、モデル化され並行プログラミング言語で記述された第１並行プログラムが格納されている。この第１並行プログラムには、バグが存在する可能性がある。第１並行プログラムを記述する並行プログラミング言語には、例えば以下のものがある。

（ａ）Ｃｏｎｃｕｒｒｅｎｔ　ＰＡＳＣＡＬ

（ｂ）ＡＤＡ

（ｃ）ＧＨＣ

（ｄ）Ｍｏｄｕｌａ３

（ｅ）Ｏｃｃａｍ
（ｆ）ｃｏｏＣ
テスト実行装置１５及びデバッグ装置１６は、それぞれ、ＨＳＰファイル記憶部１４に記憶された超遅次プログラムのテスト・デバッグを行う。
【００６３】非決定性導入装置１７は、第１並行プログラムを超遅次プログラムに変換する際に、良い非決定性を導入する。
【００６４】並行化装置１８は、ＨＳＰファイル記憶部１４に記憶された超遅次プログラムのテスト・デバッグ情報に基づいて、超遅次プログラムを並行化して、第２並行プログラムを生成する。この第２並行プログラムは、第２ＣＰファイル記憶部１９に記憶される。
【００６５】図３は、実施例１に係る並行プログラム作成方法の概略手順を示すフローチャートである。
【００６６】（１）　ステップＡ１：モデル化
対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、並行システムの各プロセス構造を決定する。更に、該各プロセス内を並行プログラム等を用いたプログラミングにより、並行構造を有する並行プログラムをソースプログラムとして記述する。なお、「並行構造を有する」としたのは、一般に並行プログラムは、その全てが並行構造で構成されているわけではなく、遅次性を用いたモデル化の方がより自然である場合には、その部分は遅次構造である場合があるからである。なお、このソースプログラムには、バグが潜在的に存在する可能性がある。
【００６７】（２）　ステップＡ２：遅次化
デフォルト遅次化によって、第１並行プログラムを遅次構造の超遅次プログラムに変換する。本実施例では、メタレベルにおいて遅次性を導入する。ここで、メタレベルとはソースプログラム（すなわち、第１並行プログラム）そのもののレベルではなく、ソースプログラムの実行を管理するレベルをいう。例えば、並行プログラムで記述されたソースプログラムを、それとは別に管理されるスケジューラによって遅次的に実行することを保障したプログラムのソースプログラムに変換する。
【００６８】（３）　ステップＡ３：超遅次プログラムのテスト・デバッグ
超遅次プログラムのテスト・デバッグを行う。テスト実行装置により超遅次プログラムをテスト実行した結果に基づいて、デバッグ装置により超遅次プログラムからバグを除去する。ここでのテスト・デバッグは、遅次プログラムにおける通常のテスト・デバッグ方法と同様に行うことができる。超遅次プログラムが正常に動作することが保障されるまで、テスト・デバッグを行う。
【００６９】（４）　ステップＡ４：超遅次プログラムに対する良い非決定性の導入
テスト・デバッグが行われた超遅次プログラムに対して、良い非決定性に関する情報（並行性に関する情報）を与える。これにより超遅次プログラムは、非決定性に関する情報を一部に持つことで、部分超遅次プログラムとなる。良い非決定性に関する情報の導入方法は後述する。
【００７０】（５）　ステップＡ５：部分超遅次プログラムのテスト・デバッグ
ステップＡ４で得られた部分超遅次プログラムに対して、テスト・デバッグを行う。すなわち、ステップＡ４で良い非決定性に関する情報が導入された超遅次プログラムをテスト・デバッグする。
【００７１】（６）　ステップＡ６：部分超遅次プログラムに対する良い非決定性の導入・拡大
ステップＡ５でテスト・デバッグが行われた部分超遅次プログラムに対して、良い非決定性に関する情報を追加する。ステップＡ５～ステップＡ６を所定の回数繰り返し、良い非決定性を徐々に拡大していく。
【００７２】（７）　ステップＡ７：並行化コンパイル
良い非決定性に関する情報が導入された部分超遅次プログラムのうち、無害な非決定性部分を抽出し、その部分を並行化することで、部分超遅次プログラム全体を並行プログラム（すなわち、第２並行プログラム）に復元する。すなわち、並行化に関する情報が導入されなかった部分に関しては、デフォルト遅次化で与えられた遅次性を並行プログラム中に反映させて（例えば、ソースプログラム自体に埋め込む）、メタレベルのデフォルト遅次性は解除する。
【００７３】図４～図７を参照して、本発明をより具体的に説明する。
【００７４】図４は、簡単な並行プログラムの一例を示す図である。
【００７５】図４の並行プログラムは、プロセスＰ１とプロセスＰ２とから構成されている。これらのプロセスは、並行プログラムのソースコードがコンパイルされることにより生成された実行モジュールがコンピュータ上で実行された時に初めて実体化するものであり、プロセスＰ１とプロセスＰ２とにそれぞれ対応する並行プログラムが必ずしも物理的に別の記憶媒体に記憶されている必要はない。また、メモリＭはここでは共有メモリ(shared memory) を表し、並行プログラムのアクセス命令によって書き込み／読み込み等のアクセスを行うことができる。図４において実線矢印はプロセスＰ１、Ｐ２が実行された時に共有メモリＭとの間でアクセスが行われることを示す。
【００７６】ＣＰファイル記憶部１１に格納された第１並行プログラムは、ユーザによる入力装置４からの指示により引き出され、遅次化装置１２に入力される。遅次化装置１２に入力された第１並行プログラムは、遅次化ルール記憶部１３に格納された遅次化ルールに従って、メタレベルにおいてデフォルト遅次性が導入されることにより、超遅次プログラム（ＨＳＰ）に変換され、ＨＳ

Pファイル記憶部１４に記録される。

【００７７】遂次化ルールの代表的なものには以下のものがある。

（ａ）プロセスに優先度を導入するルール。

（ｂ）プロセス内の処理単位（オブジェクト指向におけるメソッド）に優先度を導入するルール。

（ｃ）具体的な実行ログに基づく遂次化ルール。

（ｄ）メッセージの到着先の実行を優先する遂次化ルール。

（ｅ）メッセージの送信元の実行を優先する遂次化ルール。

【００７８】図４の下方に記載した″Ｐ１＞＞Ｐ２″が図４の並行プログラムに対する遂次化ルールの一例であり、二つのプロセスＰ１、Ｐ２に対してＰ１をＰ２より優先的に動作させるという優先度を与えることを表している。これは上記（ａ）のルールに相当する。この遂次化ルールは、例えば第１並行プログラムの先頭で宣言しておき、プログラム本体とともにコンパイルすることにより並行プログラムに導入するようにしても良いし、並行プログラムとは別にファイルに記述しておき、並行プログラムが実行時にオペレーティングシステムやスケジューラが解釈することにより導入するようにしても良い。なお、本実施例では遂次化ルールを″＞＞″で示したが、これに限らず任意の記号を用いることができる。

【００７９】図５は、メタレベルにおいてデフォルト遂次性が導入された超遂次プログラムＨＳＰの概念図であり、プロセスＰ１とプロセスＰ２とはスケジューラＳによって管理されていることを示しており、図５において、破線矢印は図４に示した遂次化ルール（″Ｐ１＞＞Ｐ２″）に従って、スケジューラＳがプロセスＰ１を実行後、プロセスＰ２を実行することを示す。なお、この超遂次プログラムＨＳＰの概念図は、図５の下方に記載した式、

ＨＳＰ＝Ｐ１｜Ｐ２｜Ｓ

のように記述するものとする。これは、超遂次プログラムＨＳＰはプロセスＰ１とプロセスＰ２とスケジューラＳとから構成されていることを示す。ここで、遂次化ルールはスケジューラのスケジューリング規則に対応する。

【００８０】ユーザは、この超遂次プログラムＨＳＰを出力装置５によって見ることができる。この超遂次プログラムＨＳＰは、ユーザによる入力装置４からの指示によってテスト実行装置１５に入力され、テスト実行が行われる。テスト実行装置１５は、テスト実行の結果（実行ログ）を出力装置５に提示する。ユーザは、このテスト実行の結果に基づいて、入力装置４をデバッグ装置１６として用いて、超遂次プログラムＨＳＰに対し所定のテスト・デバッグを行うことができる。具体的なテスト・デバッグ技術には、

（ａ）ソースコードのトレーサ

（ｂ）ブレークポイント

（ｃ）アニメーション

等がある。

【００８１】図６は、この時のテスト・デバッグのイメージを表す図である。図６において、出力装置５上のデバッグ画面６０には、各種ウィンドウ６１～６５がオープンされ、各種情報が表示されており、これらウィンドウ６１～６５は、適宜オープン・クローズすることが可能である。なお、ここでのテスト・デバッグは、基本的には公知のデバッグ装置を用いることができ、具体的には、ＵＮＩＸワークステーション上のｄｂｘｔｏｏｌ等が知られている。

【００８２】ユーザは、超遂次プログラムＨＳＰに対して所定のテスト・デバッグを行った後、入力装置４から再度テスト実行の指示を与える。これにより、テスト・デバッグの行われた超遂次プログラムＨＳＰは、テスト実行装置１５に入力され、再度テスト実行が行われる。このテスト・デバッグは、超遂次プログラムＨＳＰが正常に動作することを確認するまで繰り返し行われる。テスト・デバッグにより正常に動作することを確認した時点で、非決定性導入装置１７により超遂次プログラムに対して良い非決定性に関する情報を部分的に導入していく。非決定性導入装置１７より導入された良い非決定性に関する情報は、超遂次プログラムＨＳＰに反映され、ＨＳＰファイル記憶部１４に記録される。なお、非決定性導入装置１７については後述する。

【００８３】図７は、超遂次プログラムに良い非決定性に関する情報が導入された状態の一例を示す図である。ここで、並行プログラムの各プロセスの実行単位を「セクション」と呼ぶと、図７はセクションＳ１とセクションＳ２とに分けられたプロセスＰ１と、セクションＳ３とセクションＳ４とに分けられたプロセスＰ２について、セクションＳ２とセクションＳ３の優先度を同じものとすることにより、非決定性が導入された状態を示す。

【００８４】より具体的には、実行単位毎に異なる優先度を与えている遂次化ルールに対し所定の部分に良い非決定性に関する情報を与える（例えば、優先度を同じにしたいプロセスをマウスで指定する）ことにより、当該部分については優先度を同じとして、並行に実行可能にする。図７の例では、Ｓ１～Ｓ４の４つのセクションのうち、「Ｓ２である″ｗｒｉｔｅ１″とＳ３である″ｒｅａｄ２″の優先度を同じにする」という情報（Ｓ２＝Ｓ３）を良い非決定性として導入している。すなわち、優先度が同じプロセスは、どちらが先に実行されてもかまわないので、非決定性を持つ。

【００８５】次に、非決定性導入装置１７により良い非決定性に関する情報が導入された超遂次プログラム（部分超遂次プログラムＰＨＳＰ）は、ユーザからの指示に従ってテスト実行装置１５によりテスト実行が行われ、

デバッグ装置１６によりテスト・デバッグが行われる。この場合、部分超遅次プログラムＰＨＳＰの振る舞いは、良い非決定性に関する情報の導入された部分については非決定的な振る舞いをするので、その振る舞い全てについてテスト・デバッグを行うことが好ましい。このようにして、テスト・デバッグ及び良い非決定性に関する情報の導入を繰り返し、良い非決定性に関する情報を徐々に付加していく。

【００８６】良い非決定性に関する情報がインクリメンタル(incremental) に導入されて得られた部分超遅次プログラムＰＨＳＰは、ユーザからの指示により並行化装置１８に入力される。並行化装置１８は、部分超遅次プログラムＰＨＳＰのうち無害な非決定性部分を抽出し、部分超遅次プログラムＰＨＳＰ全体を並行化する。即ち、並行化装置１８は、導入された良い非決定性と無害な非決定性に関してデフォルト遅次性を解除し、並行プログラムＣＰ（第２並行プログラム）としてファイルに記録する。ここで、良い非決定性と無害な非決定性以外はデフォルト遅次化で与えられた遅次性が並行プログラムＣＰに反映されなければならない。ユーザは、この第２並行プログラムを出力装置５によって見ることができるとともに、最終的なテスト・デバッグを行うことができる。

【００８７】実施例１は、本発明の基本的な実施例を示したが、以下、更に詳細な実施例を実施例を説明する。但し、以下の実施例においては、実施例１と共通の部分や相対応する部分については同一符号を付して説明を簡略化するか又は省略し、相違点を中心に説明する。

【００８８】（実施例２）実施例２では、実施例１と同様に、次のような並行プログラムを対象とする。並行プログラムは複数のプロセスから構成される。並行プログラムは、共有メモリ型のマルチプロセッサで実行される。各プロセス毎にプロセッサ（ＣＰＵ）が割り当てられる。各プロセスの同期は、同期基本命令と共有メモリ型で実現される。

【００８９】上記の並行プログラムに本発明を適用した実施例を説明する。図８は、実施例２に係る並行プログラム作成支援装置の概略構成を示すブロック図であり、図９は、実施例２に係る並行プログラム作成方法の概略手順を示すフローチャートである。図８が、図２と異なる点は、セクション設定装置７を更に具備し、テスト実行装置１５とデバッグ装置１６とを、具体的に、テスト実行装置１５、修正装置９、及び解析装置１０とし、解析装置１０で解析された解析情報を記憶する解析情報記憶部２０を有する点である。解析装置１０では、超遅次プログラムを解析し、解析情報として後述する先行制約を抽出する。他の構成部分は実施例１と同様であるので、説明を省略する。なお、図８には、並行化装置１８で、超遅次プログラムの並行化を行う際に参照される並行化ルールを記憶する並行化ルール記憶部２１を入れている。

【００９０】セクション設定装置７は、第１並行プログラムの各プロセスをいくつかのセクション（プログラム単位）に分割する。

【００９１】修正装置９は、テスト実行装置１５によるテストの結果バグがあれば、修正を行う。

【００９２】テスト実行装置１５は、実施例１と同様に、超遅次プログラムのテストを行う。

【００９３】解析情報記憶部２０は、解析装置１０で解析された情報を記憶する。

【００９４】実施例２の動作を図９のフローチャートを参照して説明する。なお、図９において、図３のフローチャートと同一動作には同一の符号を付す。

【００９５】（１）　ステップＡ１：モデル化

対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、並行システムの各プロセス構造を決定する。更に、該各プロセス内を並行プログラム等を用いたプログラミングにより、並行構造を有する並行プログラムをソースプログラムとして記述する。このソースプログラムには、バグが潜在的に存在する可能性がある。

【００９６】（２）　ステップＢ１：セクションの設定

セクション設定装置７により、設計者が第１並行プログラムの各プロセスをいくつかのセクション（単位）に分割する。ここで、第１並行プログラム中の同期命令は自動的に単独のセクションとする。ここで、セクションはプロセスの処理の単位であり、以下のステップでは、セクションを単位として、遅次化及び並行化を行う。この場合において、設計者によるセクション設定がなくても構わない。この場合は、同期命令で区切られる区間が自動的にセクションになる。

【００９７】セクションの設定は、プロセスのソースコードを分割し、分割された各区間にセクションＩＤを設定することで実現できる。ソースコード分割の一例としては、図１０に示すように、区切りポイントを挿入し、区切りポイントから次の区切りポイントまでの処理をセクションとする方法がある。ここで、上記のように同期命令の前後には自動的に区切りポイントが挿入される。

【００９８】（３）　ステップＡ２：遅次化

遅次化ルールに基づき、遅次化装置により、第１並行プログラムを遅次化する。遅次化ルールの一例としては、プロセスに優先度を導入する方式がある。この優先度に基づいて実行すれば、実行時の非決定性は存在しないので、超遅次プログラムとみなすことができる。このようにして並行構造に関するプログラム情報を持ちながら遅次化されたプログラムを超遅次プログラムとする。遅次化方式の一例としては、「プロセスの優先度」に基づく方式がある。この方式によれば、プロセスに予め固定の優先度を設定し、優先度が高いプロセスのセクションの実行を優先することにより、非決定性のない遅次的な実

行順序が得られる。別の逐次化方式の例としては、図１１に示すような「同期命令のwait側の実行を優先する方式（逐次化ルール１）」や「同期命令のsend側の実行を優先する方式（逐次ルール２）」等がある。ここで、超逐次プログラムは、３種のプログラム情報、すなわちセクション情報と、プログラム構造情報と、逐次化情報とから構成される。セクション情報は、セクションの識別子（ＩＤ）とセクションの属するソースコードの情報である。プログラム構造情報は、オリジナルの並行プログラムにおける、プロセス毎のセクションの実行順序情報、更に、異なるプロセスのセクション間のデータ依存関係の情報である。逐次化情報は、逐次化によるグローバルなセクションの実行順序情報であり、同時によい並行性の関する情報も持つ。一例としては、並行システムのモデル化手法であるペトリネットで逐次化情報を表現する方法がある。

【００９９】（４）　ステップＡ３：超逐次プログラムに対するテスト・デバッグ

超逐次プログラムに対して、テスト実行装置１５でテストを行い、バグがある場合は、修正装置９でデバッグ／修正を行う。また、プログラムの修正が同期命令等の並行構造の変更にも及ぶ場合は、ステップＡ１に戻り、プログラム作成装置８でモデル化を行い、再度セクションの設定及び逐次化を行う。ここで、プログラム６は逐次化されているので、テスト・デバッグは逐次プログラムなみに容易になる。

【０１００】（５）　ステップＡ４：超逐次プログラムに対する良い非決定性の導入

非決定性導入装置１７により、出力装置５に超逐次プログラムの構造が示される。設計者はこの構造を見ながら非決定性導入装置１７により良い非決定性による並行性を明示的に導入できる。この時、解析装置１０により抽出された超逐次プログラムのプログラム情報により、設計者による良い非決定性の導入を支援する。良い非決定性の導入の必要がなければ、ステップＢ２に進む。良い非決定性の導入方式として、例えば、超逐次プログラムの逐次化情報がペトリネットで表現されている場合には、プログラム構造を保存する範囲でのペトリネットの書換によって良い非決定性を導入する。

【０１０１】（６）　ステップＡ５：テスト・デバッグ

良い非決定性が導入された超逐次プログラムに対して、ステップＡ３と同様に、テスト実行装置１５でテストを行い、バグがある場合は、修正装置９でテスト・デバッグを行う。また、プログラムの修正が同期命令等の並行構造の変更にも及ぶ場合は、ステップＡ１に戻り、プログラム作成装置で修正を行い、再度セクションの設定及び逐次化を行う。ここで、プログラム６で並行化が導入された部分に関しては、並行に実行を行う。この場合において、超逐次プログラムの逐次化情報がペトリネットで表現されている場合には、ペトリネットのシミュレータを用いて、トークン(token) のあるプレースに対応するセクションを実行することで、超逐次プログラムのテスト実行が実現できる。

【０１０２】（７）　ステップＡ６：部分超逐次プログラムに対する良い非決定性の導入・拡大

ステップＡ５でテスト・デバッグが行われた超逐次プログラムに対して、良い非決定性に関する情報を追加する。ステップＡ５～ステップＡ６を所定の回数繰り返し、良い非決定性を徐々に拡大していく。更に良い非決定性を導入する必要があれば、ステップＡ５と同様に非決定性導入装置１７で並行性を追加し、ステップＡ６に戻る。必要がなければ、ステップＢ２に進む。

【０１０３】（８）　ステップＢ２：自動並行化

設計者による良い非決定性の導入が終わった超逐次プログラムに対して、解析装置１０により抽出された超逐次プログラムの解析情報１２により、並行可能な部分を自動抽出し、並行化装置１８で並行化ルールを用いて超逐次プログラムの並行性を自動拡大する。並行化ルールは、例えば、以下のようになっている。プログラム情報から、解析装置１０によってデータ依存関係及び制御依存関係に基づく先行制約を抽出し、先行制約のないセクションに関しては並行化できる。この並行化処理は、予め定められた並行化ルールを適用することにより行うことができる。並行化ルールの一例としては、図１２に示すようなペトリネットによる逐次化情報の書換ルールがある。データ依存関係及び制御依存関係に関しては、公知である。

【０１０４】（９）　ステップＢ３：並行プログラムの作成

並行化装置１８において、自動的に並行性を拡大した超逐次プログラムに対して、超逐次プログラムのプログラム情報をソースコードに反映させた並行プログラム１５を生成する。

【０１０５】通常、逐次プログラムに比べて、並行プログラムのテスト・デバッグは非常に困難な作業である。これは、並行性によるプログラムの非決定により、あるタイミングによっては、プログラムが設計者の意図しない挙動を示すからである。並行プログラムのテスト・デバッグでは、設計者の意図しない並行性（悪い非決定性）によるバグを１つ１つテストで発見して取り除いている。しかし、この方法では、バグを完全に取り除くのは非常に困難であり、更に労力を要する。

【０１０６】本発明の超逐次プログラミングでは、並行プログラムをまず逐次化し、それに対して設計者が意図する並行性を徐々に導入し、最終的に自動的に並行化できる部分を並行化し、並行プログラムを復元する。

【０１０７】すなわち、本発明の超逐次プログラミングは、並行プログラムから悪い非決定性を除くのではなく、逐次プログラムに良い非決定性を導入する。このように、超逐次プログラミングでは、逐次プログラムから

ボトムアップ的に並行プログラムを作成するので、予期せぬタイミングで発生するバグが入る余地がなく、非常に信頼性の高いプログラムを作成することができる。また、テスト・デバッグがはるかに容易になる。逐次プログラミングでは、逐次化による性能劣化等が懸念されるが、スーパーコンピュータ等の分野におけるＦＯＲＴＲＡＮ等の逐次プログラムの自動並行化技術が利用できるので、多くの場合では実用上問題がない。実施例２の基本的な構成及び動作を説明したが、以下に、実施例２の具体例を示す。

（ａ）　第１具体例

図１３は、並行プログラムの例を示す。ここで、Ｐ１とＰ２は並行に動くプロセスである。また、Ｐ１とＰ２は、共有メモリＭをアクセスしている。

【０１０８】（１）　ステップＡ１：モデル化

並行プログラムＰを図１４のように記述する。

【０１０９】（２）　ステップＢ１：セクションの設定

この場合は、各命令が１つのセクションを形成すると考える。簡単のため、セクションＩＤを命令そのものとする。

【０１１０】（３）　ステップＡ２：逐次化

プロセス優先度を導入することによる逐次化を行う。具体的には、Ｐ１＞＞Ｐ２（Ｐ１はＰ２より優先する）とする。この時、逐次化されたセクションの実行順序は、以下のようになる。

ｉｎｉｔ１→ｒｅａｄ１→ｗｒｉｔｅ１→ｒｅａｄ２→ｗｒｉｔｅ２

逐次化により生成された超逐次プログラムは、それぞれ図１５に示すように、セクション情報、プログラム構造情報、超逐次化情報から構成される。ここで、逐次化情報は、上記の実行順序をペトリネットで表現したものである。

【０１１１】（４）　ステップＡ３：超逐次プログラムに対するテスト・デバッグ

超逐次プログラムを実行し、バグがあればセクションの各セクションのソースコード、或いは、オリジナルの並行プログラムを修正する。ここでは、バグはなかったとする。

【０１１２】（５）　ステップＡ４：超逐次プログラムに対する良い非決定性の導入

逐次化情報のペトリネットを表示装置で表示し、逐次関係を切断することにより、並行化を行う。ここでは、ｗｒｉｔｅ１とｒｅａｄ２の逐次関係を切断することにする（図１６（ａ））。ｗｒｉｔｅ１とｒｅａｄ２はプログラム構造情報での実行順序関係はないので、切断可能である。ここで、どの逐次関係を切断すべきかに関するガイダンス情報を後述する解析情報２０に基づき提供することもできる。

【０１１３】（６）　ステップＡ５：テスト・デバッグ

並行性の導入された超逐次プログラムをテスト実行する。ここでは、

ｉｎｉｔ１→ｒｅａｄ１→ｗｒｉｔｅ１→ｒｅａｄ２→ｗｒｉｔｅ２

ｉｎｉｔ１→ｒｅａｄ１→ｒｅａｄ２→ｗｒｉｔｅ１→ｗｒｉｔｅ２

のような実行が可能である。実行の結果、バグがあれば修正する。ここでは、バグはなかったとする。

【０１１４】（７）　ステップＢ２：自動並行化

超逐次プログラムの逐次化情報とプログラム情報のデータ依存関係から、解析情報２０により、セクション間の先行制約が得られる。この先行制約が解析情報２０である。先行制約とは、データ依存関係にあるセクション間の実行順序の制約である。すなわち、データ依存関係にあるセクション間は、実行順序によって計算結果が変わり得るので、逐次化情報で定められた順序を保持する必要がある。ここでの先行制約は以下の３つである。

ｉｎｉｔ１→ｒｅａｄ２

ｉｎｉｔ１→ｗｒｉｔｅ２

ｒｅａｄ１→ｗｒｉｔｅ２

例えば、Ｐ１がｒｅａｄ１で読み込む値は、Ｐ２のｗｒｉｔｅ２がｒｅａｄ１の前で起こるか後で起こるかによって影響を受ける。逐次情報では、ｒｅａｄ１→ｗｒｉｔｅ２であるので、これが先行制約となる。以上の先行制約がないセクションに関しては、並行化が可能であり、並行化ルールにより自動並行化できる。ここでは、ｒｅａｄ１とｒｅａｄ２には先行制約がないため、並行化ルール１（図１２）を適用できた（図１６（ｂ））。

【０１１５】（８）　ステップＢ３：並行プログラムの作成

自動並行化を行った逐次プログラミングから、並行プログラムのソースコードを生成する。この例では、図１０における逐次化情報１と逐次化情報２を実現する同期命令（ｓｅｎｄ、ｗａｉｔ）をソースコードに埋め込む。その他の逐次化情報は、オリジナルの並行プログラムが持っている逐次化情報（実行順序）である。変換されたプログラムは図１７のようになる。この並行プログラムは、図１３のような構造の並列計算機上で実行される。

（ｂ）　第２具体例

第２具体例では、並行プログラムが同期命令、ループ構造、条件分岐を持つ場合の超逐次プログラミングの手順を示す。

【０１１６】（１）　ステップＡ１：モデル化及びステップＢ２：セクションの設定は省略し、図１８のようなセクションの実行順序（プログラム構造情報）を持つ並行プログラムが与えられたとする。ここで、同期命令（ｓｅｎｄ、ｗａｉｔ）とデータ依存関係（Ｓ１３とＳ２２）、ループ構造及び条件分岐があることに注目されたい。

【０１１７】（２）　ステップＡ２：逐次化

プロセスに優先度（Ｐ１がＰ２より優先度が高い；すな

わち、Ｐ１＞＞Ｐ２）を導入することによる遂次化を行う。遂次化の結果は図１９のようになる。すなわち、超遂次プログラムは、それぞれ図１８及び図１９のプログラム構造情報と遂次化情報を持つ（セクション情報は省略）。ここで、同期命令によって実行されるプロセスが切り替わっていることに注目されたい。例えば、ｗａｉｔ命令によって、Ｐ１のＳ１２の実行の後にＰ２のＳ２１が実行されている。また、ループ及び分岐構造もペトリネットで表現されている。

【０１１８】（３）　ステップＡ３：超遂次プログラムに対するテスト・デバッグ

超遂次プログラムを実行し、バグがあればセクションの各セクションのソースコード、或いは、オリジナルの並行プログラムを修正する。ここでは、バグはなかったとする。

【０１１９】（４）　ステップＡ４：超遂次プログラムに対する良い非決定性の導入

遂次化情報のペトリネットを表示装置で表示し、遂次関係を切断することにより、並行化を行う。この例では、明示的には、良い非決定性を入れなかったとする。

【０１２０】（５）　ステップＢ２：自動並行化

超遂次プログラムの遂次化情報とプログラム情報のデータ依存関係から、セクション間の先行制約が得られる。ここでは、先行制約としては、Ｓ２２→Ｓ１３だけである。ここで、ループにおいては、先行関係にもループが生じることがある。ここでは、１回のループにおける先行関係のみに注目する。先行制約がないセクションに関しては、並行化が可能であり、並行化ルールにより自動並行化できる。この例では、先行関係はＳ２２→Ｓ１３だけであり、図１２の並行化ルール１と並行化ルール２により、最終的に図２０が生成できる。

【０１２１】（６）　ステップＢ３：並行プログラムの変換

第１具体例と同様に、残った遂次化情報に関して同期命令を埋め込むことによって並行プログラムが生成できる。

【０１２２】（実施例３）図２１は実施例３に係る並行プログラム作成支援装置の概略構成を示す図であり、図２２は、本実施例に係る並行プログラム作成方法の概略手順を示すフローチャートである。

【０１２３】本実施例では、テスト実行装置４０１とテストケース記憶部４０２によって遂次化装置１２が構成される。この遂次化装置１２では、今までの実施例と異なり特別に遂次化ルールはなく、遂次化はランダムに行われる。すなわち、ランダムに実行したときのログを超遂次プログラムとみなす。以下、本実施例を具体的に説明する。

【０１２４】本実施例における並行プログラムの作成は、以下のような手順によって実現される。ここで、第１並行プログラムとして全く独立に動く２つのプロセスＰ１とＰ２から構成される簡単な並行プログラムを考える。

【０１２５】（１）　ステップＡ１：モデル化

対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、各プロセス構造を決定する。更に、該プロセス内を並行プログラム等を用いたプログラミングにより、並行構造を有する並行プログラムをソースプログラムとして記述する。このソースプログラムには、バグが潜在的に存在する可能性がある。ここでは、２つのプロセスＰ１とＰ２を図２３に示すようにプログラミングする。

【０１２６】（２）　ステップＣ１：テスト

テスト実行装置４０１により、テストケース記憶部４０２からテストケースを引き出し、ＣＰファイル記憶部１１からの第１並行プログラムを実行し、実行結果を出力装置５に表示してランダムにテストを行う。ここでは、実行ログが、

ｌｏｇ１＝ｊｏｂ１１→ｊｏｂ１２→ｊｏｂ２１→ｊｏｂ２２

であったとする。

【０１２７】（３）　ステップＣ２：バグ判定

ステップＣ１のテストの結果、バグがなければステップＣ４に進み、バグがあれば、その実行ログを実行ログ記憶部４０３に保存してステップＣ３に進む。ここで、もしバグがあればステップＣ３に進むが、ここではバグがなかったとしてステップＣ４に進む。

【０１２８】（４）　ステップＣ３：テスト・デバッグ

デバッグ装置１６により、実行ログ記憶部４０３に保存された実行ログに基づくバグの発見を出力装置５を用いて行い、ＣＰファイル記憶部１１に格納されている第１並行プログラムを修正してバグを除去する。テスト・デバッグが完了したら、ステップＣ１に戻る。

【０１２９】（５）　ステップＣ４：実行ログの蓄積

ステップＣ１のテストの結果、ステップＣ２でバグがないと判定されれば、実行ログｌｏｇ１を実行ログデータベース４０４に蓄積する。実行ログは、超遂次的な実行系列であり、超遂次プログラムの特殊な形態とみなすことができる。

【０１３０】（６）　ステップＣ５：残テストケースの有無判定

テストケース記憶部４０２にテストケースが残っているかどうかを判定し、残っていればステップＣ１に戻ってテストを続行し、テストケースが残っていなければステップＡ７に進む。本ステップでは、別のテストケースも試みるためにステップＣ１に戻ってテストを続行する。この２回目のテストで、次に示す実行ログｌｏｇ２が実行ログデータベース４０４に保存されたとする。

ｌｏｇ２＝ｊｏｂ１１→ｊｏｂ２１→ｊｏｂ１２→ｊｏｂ２２

（７）　ステップＡ７：並行プログラムの生成

ＣＰファイル記憶部１１に格納されている第１並行プログラムと実行ログデータベース４０４に格納されている実行ログから、テストで通過したパスだけしか実行しないような並行プログラムを並行化装置１８により生成し、第２ＣＰファイル記憶部１９に格納する。

【０１３１】本ステップでは、実行ログデータベース４０４に保存された２つの実行ログｌｏｇ１、ｌｏｇ２だけを実行するような並行プログラムを以下の手順で生成する。

（Ａ）全ての実行ログをマージしてグローバル状態遷移システムＧＴＳ(Global Transition System)を生成する（図２４（ａ））。

（Ｂ）ＧＴＳを２つのプロセスＰ１、Ｐ２に射影する。この時、相手のプロセスの挙動を認識し、同期を取るための同期命令（ｊｏｂ１１－ｏｋ、ｊｏｂ２１－ｏｋ、ｊｏｂ１２－ｏｋ、ｊｏｂ２２－ｏｋ）が埋め込まれる。生成されたプログラムをＰ１′、Ｐ２′とする（図２４（ｂ））。

（Ｃ）Ｐ１′、Ｐ２′の同期命令には冗長性があるため、この冗長性を除去する。この冗長性を除去した後の最終的なプロセスをＰ１″、Ｐ２″とする（図２４（ｃ））。

【０１３２】上記の手順により、２つのプロセスＰ１″、Ｐ２″から構成される並行プログラムＣＰ″が生成できる。この並行プログラムＣＰ″では、テストで正しく動くことを確認した実行ｌｏｇ１、ｌｏｇ２は可能だが、テストしてないケース、例えば

ｌｏｇ３＝ｊｏｂ２１→ｊｏｂ１１→ｊｏｂ１２→ｊｏｂ２２

は実行されることはない。

【０１３３】上記のように、テストしてないケースは実行されることはないので、非常に安全なプログラムであるといえる。

【０１３４】実施例３によれば、逐次プログラミングと同程度の困難さで並行プログラムのプログラミングができ、ユーザはテストを効率的にできるばかりではなく、テストしない部分は動かないので、非常に高い信頼性が達成できる副次的効果も期待できる。

【０１３５】（実施例４）図２５は、実施例４における並行プログラム作成方法の概略手順を示すフローチャートである。

【０１３６】（１）　ステップＡ１：モデル化

（２）　ステップＡ２：逐次化

（３）　ステップＡ３：超逐次プログラムのテスト・デバッグ

以上のステップＡ１～Ａ３は、図３に示したステップＡ１～Ａ３と全く同様であるため、説明を省略する。

【０１３７】（４）　ステップＡ４：超逐次プログラムに対する良い非決定性部分の導入

ステップＡ３でテスト・デバッグが行われた超逐次プログラムに対して、ユーザが非決定性導入装置１７により、非決定的に動作する部分（これを「良い非決定性部分」という）を指定する。非決定性導入装置１７により指定された良い非決定性部分は、超逐次プログラムに良い非決定性に関する情報（並行性に関する情報）として反映される。これにより超逐次プログラムは、良い非決定性部分を持つ部分超逐次プログラムとなる。良い非決定性部分の指定方法は後述する。

【０１３８】（５）　ステップＡ５：部分超逐次プログラムのテスト・デバッグ

ステップＡ４で得られた部分超逐次プログラムに対して、テスト・デバッグを行う。すなわち、ステップＡ４で良い非決定性部分が導入された部分超逐次プログラムをテスト・デバッグする。ここでは、部分超逐次プログラムを実行形式に変換して実行する際に、ステップＡ４において導入された良い非決定性部分のみが並行構造の実行形式に、導入されなかった部分は元の逐次構造のまま実行形式に変換され、実行される。そして、この実行結果に基づいて、超逐次プログラムに対してテスト・デバッグする。変換された実行形式のプログラムが正しく動作しない場合は、変換前の逐次プログラムにおいて導入した良い非決定性部分は本来、非決定的に動作させるべき部分ではないとみなして、並行化の解除を行う。

（６）　ステップＡ６：超逐次プログラムにおける良い非決定性部分の拡大

ステップＡ５でテスト・デバッグが行われた部分超逐次プログラムに対して、良い非決定性に関する情報を追加する。ステップＡ４～ステップＡ６を所定の回数繰り返し、良い非決定性部分を徐々に拡大していく。

【０１３９】（７）　ステップＡ７：並行化コンパイルメタレベルのデフォルト逐次性で管理されている部分超逐次プログラムのデフォルト逐次性を解除する（例えば、ソースプログラム自体に逐次情報及び並行情報を直接埋め込む）。また、この時、部分超逐次プログラムのうちの良い非決定性に関する情報が導入されなかった部分について、可能な限り、無害な非決定性部分を抽出し、その部分を並行化することを試みて、部分超逐次プログラムの並行性を高める。例えば、良い非決定性に関する情報が導入されなかった部分について各プロセス間の依存関係を解析し、依存関係がないと解析された部分に対応するソースプログラムの部分の逐次性を解除する。なお、無害な非決定性部分の抽出は、ステップＡ２で逐次化する際に解析された並行情報を参照することにより行われる。

【０１４０】次に、本実施例に係る並行プログラム作成方法及び作成支援装置の詳細について説明する。

【０１４１】本実施例は、出力装置５上に表示された超逐次グラフを用いることにより、非決定性部分の導入を、視覚的に行うことができるようにしたことを特徴とする。超逐次グラフとは、並行構造を有する並行プログ

ラムに対してデフォルト逐次性を導入した超逐次プログラムのプロセス処理順序を表した遷移グラフをいうものとする。この超逐次グラフの表示は、逐次化情報に基づいて行われる。逐次化情報は、逐次化装置１２により並行プログラムが超逐次プログラムに変換される際に生成される。

【０１４２】より具体的には、逐次化情報とは、所定のフィールドを有するプロセステーブルと呼ばれるデータ列群である。図２６は、プロセステーブルの構造を示す図である。

【０１４３】図２６において、名前フィールドＦ１は、個々のプロセスを識別するための名前を保持する。ポインタフィールドＦ２は、当該プロセスから呼び出されるプロセス（以下「被呼出プロセス」という）の名前リストを格納する被呼出プロセスリストテーブル（図示しない）のポインタを保持する。この被呼出プロセスリストテーブルは、別に設けられている。優先順位フィールドＦ３は、プロセスの優先順位を保持する。このプロセスの優先順位は、もとの並行構造を有するプログラムに基づき、ある処理時点において複数のプロセスが実行可能な場合に、逐次化装置１２によって決定され、その結果が当該フィールドに記述される。優先順位バッファフィールドＦ３は、後述するように非決定性導入装置１７により、その値が変更される。このため、優先順位バッファフィールドＦ４は、変更前の優先順位を保持することにより、変更前の状態に戻すことを可能とする。グループ化情報フィールドＦ５は、特定のノード群がグループ化された場合に、各グループ間を識別する情報を保持し、グループ化されたノード群に対応するプロセス群のうち、最初に実行されるプロセスの名前が書き込まれる。

【０１４４】本実施例における逐次化装置１２について説明する。

【０１４５】図２７に示すような並行プログラムが逐次化装置１２に読み込まれたとする。逐次化装置１２は、まず並行プログラムの並行構造を解析する。図２７に示す並行プログラムは、概念的には図２８のような並行構造を有するものであると解析される。この解析は、例えば、木構造探索アルゴリズム等を用いることにより実現できる。そして、逐次化装置１２はその解析結果をプロセステーブルに書込む。具体的には、逐次化装置１２は、図２６に示したプロセステーブルの優先順位フィールドＦ３に優先順位を書込むことによってデフォルト逐次性を導入し、超逐次プログラムに変換する。すなわち、図２８において、プロセスＢの処理の後、プロセスＣとプロセスＤとのいずれが処理されるかは、その時点におけるシステム環境等により異なる。このような場合に、逐次化装置１２は、所定の規則（逐次化ルール）により優先順位を一意に決定する。なお、所定の規則とは、例えば、プロセスの読み込み順に従うとする規則や乱数により決定するという規則等があげられるが、種々の事情に応じてユーザが設定することができるものとする。なお、本実施例における超逐次プログラムは、概念的には、ソースプログラムとそれをメタレベルで管理する逐次化情報とにより構成されることになる。

【０１４６】例えば、上記の例において、所定の規則によりプロセスＣの方がプロセスＤよりも優先的に処理するものと決定されると、プロセステーブルにおけるプロセスＣの優先順位フィールドＦ３には「１」が書き込まれ、プロセステーブルにおけるプロセスＤの優先順位フィールドＦ３には「２」が書き込まれる。なお、優先順位を決定する必要がないプロセスについては、優先順位フィールドＦ３は「０」のままである。この優先順位の決定は、階層レベルが等しいプロセス間で行われる。例えば、図２８におけるプロセスＣ、Ｇ、Ｉとプロセス Ｄ、Ｈ、Ｊとは、同じレベルのプロセスであるが、その中のプロセスＥとプロセスＦとは一階層下のレベルであるとされる。

【０１４７】このようにして逐次化装置１２は、並行プログラムの並行構造を解析して逐次化情報（プロセステーブル）を生成し、これをファイルに記録する。

【０１４８】図２９は、逐次化装置１２によって並行プログラムの構造を解析した結果作成されたプロセステーブルの内容を示す図である。すなわち、図２９はプロセスＡからプロセスＫまでに対応する各レコードにおける各フィールドの内容を示す。より具体的には、プロセスＡの次に呼び出されるプロセスはプロセスＢであり、プロセスＡの優先順位は０であることを示す。また、例えば、プロセスＢでは次に呼び出されるプロセスとしてプロセスＣ又はプロセスＤであることを示しており、プロセスＣとプロセスＤとは本来非決定的であるということを意味している。そして、このプロセスＣとプロセスＤとは逐次化装置１２によりそれぞれ優先順位「１」と「２」とが割り当てられ、プロセスＣがプロセスＤよりも優先的に呼び出される（実行される）ことを意味している。

【０１４９】良い非決定性に関する情報の導入（つまり、良い非決定性部分の指定方法）について説明する。良い非決定性部分の導入は、図２に示した非決定性導入装置１７を用いたユーザによる対話的操作により行われる。すなわち、この非決定性装置１７は、出力装置５上に超逐次グラフを表示し、ユーザは、この超逐次グラフに対して入力装置４を用いて、良い非決定性部分の導入・解除を行う。

【０１５０】図３０は、超逐次グラフの一例を示す図である。図３０において、各ノードは各プロセスを表し、破線矢印で示されたアークはもとの並行プログラムの並行構造を表している。また、実線矢印は逐次化装置１２によりデフォルト逐次性が導入されることにより決定されたプロセスの逐次構造を表し、その逐次構造に従って

プロセスの実行順序が連番で付されている。つまり、もとの並行プログラムにおいては、プロセスC→G→Iとプロセス D→E→F→H→Jとは、本来並行処理されるように記述されていることを表しているが、超遅次プログラムにおいてはプロセスC→G→Iを一連に処理した後、プロセスD以降を処理することを表している。

【0151】ユーザは、このような超遅次グラフを出力装置5により視覚的に把握することができる。更に、ユーザは、並行プログラムが遅次化装置12により遅次化され、遅次化情報が生成された後は、超遅次グラフを表示する旨の指示を与えることにより、いつでも出力装置5により視覚的に把握することができる。本実施例では、超遅次グラフの表示は、非決定性導入装置17の超遅次グラフ表示制御部によって行われる。この超遅次グラフ表示制御部は、ユーザにより超遅次グラフを表示する旨の指示を受けると、所定の内蔵プログラムに従って超遅次グラフの表示を開始する。

【0152】図31は、超遅次グラフ表示制御部の構成を示す機能ブロック図である。　超遅次グラフ表示制御部は、遅次化情報記憶部31に格納されている遅次化情報に従って、各プロセスに対応するノードと、ノード間の呼び出し関係を示す並行構造及び遅次構造に対するアークを決定し、これらを画像データ生成部35により出力装置5上に表示するために必要な画像データに変換する。

【0153】遅次化情報読込部32は、遅次化情報記憶部31から遅次化情報を読み込み、並行構造解析部33及び遅次構造解析部34に送出する。

【0154】並行構造解析部33及び遅次構造解析部34は、遅次化情報を参照して並行構造を解析する。具体的には、並行構造解析部33は、ポインタフィールドF2に指示された被呼出プロセスリストテーブルを参照することにより、名前フィールドF1に記述されたプロセスによって呼出され得るプロセス名の間の呼出関係を特定する。遅次構造解析部34は、これらのフィールドに加え、更に優先順位フィールドF3に記述された優先順位を参照して遅次構造を特定する。これら並行構造解析部33及び遅次構造解析部34による解析結果は、画像データ生成部35に送出される。画像データ生成部35は、入力されたこれらの解析結果に基づいて、各プロセスに対応するノードを表示するための画像データと、該プロセス間の呼出関係を接続するための2つのアーク（つまり、並行構造に対するアーク及び遅次構造に対応するアーク）を表示するための画像データを生成し、出力装置5に送出する。出力装置5は、これらの画像データに基づいて遷移グラフ（即ち、超遅次グラフ）を表示する。

【0155】なお、上記超遅次グラフの表示に際し、並行構造解析部33により解析された並行構造の各ノード間の接続関係に、従属的に遅次構造解析部34により解析された遅次構造の各ノード間の接続関係を付加することにより画像データを生成することが望ましい。換言すれば、超遅次グラフは、もとの並行プログラムの並行構造を保持した超遅次グラフであることが望ましい。これにより、ユーザは、もとの並行プログラムが本来持つ並行構造を意識しつつ、遅次化された処理の流れ（プロセス）に対して、良い非決定性部分を導入することができる。

【0156】出力装置5に超遅次グラフが表示されると、図2の非決定性導入装置17はユーザによる良い非決定性部分の入力待ちの状態になる。（部分）超遅次プログラムのテスト・デバッグ開始当初は、非決定性部分が導入されていないので、遅次構造に対するアークは出力装置5上の全てのノードに対して設けられており、並行化情報は表示されていない。ここで、並行化情報とは、良い非決定性部分が導入されたノードに対して与えられる情報であり、良い非決定性部分が導入されたか否かにより画面上で視覚的に区別されて（例えば陰影の濃度、色分け等）表示される。また、この並行化情報は、階層化された良い非決定性部分についても視覚的に区別されて表示される。

【0157】ユーザによる良い非決定性部分の導入は、入力装置4を用いて行われる。具体的な良い非決定性の導入方法については後述するが、略説すれば、入力装置4を用いてノード間の遅次構造の接続関係（アーク）を断ち切ったり、接続したりすることにより行われ、その結果は、遅次化情報変更部36により遅次化情報記憶部31の内容が変更される。変更された内容は、再び遅次化情報読込み部により読み込まれ、出力装置5上に表示されることになる。

【0158】良い非決定性部分の導入の具体的操作例について説明する。

【0159】図32は、良い非決定性部分の導入方法を説明するための図である。

【0160】複数のプロセス間に対して良い非決定性を導入する場合、良い非決定性の導入モードにおいて、該プロセスに対応する所望のノードを続けて指定することにより行われる。良い非決定性の導入モードは、例えば、指示操作メニュー等により選択され、該モードに移行する。

【0161】図32において、プロセスEとFとの並行化を許容する場合、入力装置4により操作されるカーソルPでノードE、Fを続けて指定する。この指定されたノードは、指定されていない他のノードと視覚的に区別できるように、陰影が付される。対象とするノード全てについて指定が完了した場合、ユーザは完了した旨を非決定性導入装置17に与える。これにより、遅次化情報変更部36は、遅次化情報記憶部31内のプロセステーブルにおける対応するプロセスの優先順位フィールドF3の現在の値を優先順位バッファフィールドF4に対比

させるとともに、優先順位フィールドＦ３の値を「０」に変更する。

【０１６２】つまり、本実施例では、プロセスＥ及びＦの優先順位フィールドＦ３の値「１」及び「２」がそれぞれ優先順位バッファフィールドＦ４に対比されるとともに、それぞれの優先順位フィールドＦ３の値は「０」に変更される。遂次化情報変更部３６による遂次化情報記憶部３１の更新後、再度、出力装置５上に超遂次グラフが表示される。この時、プロセスＥとＦとの間の優先順位はないため、それに対応する遂次構造のアークは表示されない。つまり、出力装置５上では、プロセスＥとＦとは、並行に動作することが確認される。

【０１６３】図３３は、上記の場合において、良い非決定部分が導入された超遂次グラフの一例を示す図である。

【０１６４】上記のようにして良い非決定性部分が導入されたプログラム（部分超遂次プログラム）は、テスト実行装置１５により実行形式に変換され、テスト実行される。ユーザは、テスト実行装置１５による実行結果に基づいて、プログラムが正しく動作するか否かを確認する。この動作確認は、並行性に基づく全てのパスについて実行する必要がある。テスト実行装置１５が正しく動作しなかった場合、良い非決定性部分を導入した部分は、本来並行化を許容する部分ではないとみなすことができるため、ユーザは並行化を解除する指定をする。

【０１６５】並行化の解除は、非決定性部分解除モードにおいて、良い非決定性部分を導入する場合と同様に指定することにより行われる。これにより、遂次化情報変更部３６は、遂次化情報記憶部３１内のプロセステーブルにおける優先順位バッファフィールドＦ４に格納されている値を優先順位フィールドＦ３に戻す。

【０１６６】本実施例では、良い非決定性部分の導入に際し、階層レベルごとにグループ化することも可能である。例えば、プロセスＣ、Ｇ、Ｉのグループ（グループ１という）とプロセスＤ、Ｈ、Ｊのグループ（グループ２という）とは同一の階層レベルであるので、これらは互いに並行動作しうる。従って、個々のプロセスについて操作したのでは操作性に欠けるため、これらをグループ化することが有効である。このグループ化は、グループ化モードを選択することにより行われる。具体的には、グループ化モードにおいて、対象とするノードを順次選択し、完了した旨を与えることにより、対象となるノードがグループ化されて１つの新たなノードとして表示される。

【０１６７】図３４は、グループ化されたノードを持つ超遂次グラフの一例を示す図である。

【０１６８】図３４において、グループ１及び２のノードは、グループ化されていない他のノードと視覚的に区別できるように、例えば、楕円形で表示されている。更に、グループ２は、更に下位階層（プロセスＥ及びＦ）を持っているため、例えば、影付きで表示される。このような超遂次グラフに対し、ユーザは、良い非決定性部分を指定することができる。なお、このようなグループ化された超遂次グラフに対して良い非決定性部分を導入した場合であっても、先頭のプロセス（プロセスＣ及びＤ）の優先順位フィールドＦ３を変更するだけでよい。

【０１６９】超遂次グラフのノードに対して良い非決定性部分を導入する場合には、上述したように所望のノードを入力装置４により直接指定することにより行われるが、このような方法ではなく、特定のノード群を囲い込むことによって、又は指定領域が特定のノード群に掛かることによって指定されたとするようにしてもよい。なお、プロセスが多数ありそれに対応するノードが多数あるため、出力装置５上で確認することが困難な場合も想定できる。このような場合は、グループ化されたノード群のみを表示することにより対処することができる。

【０１７０】本実施例では更に、遂次化装置１２により決定されたプロセス間の優先順位を変更することもできる。つまり、優先順位変更モードにおいて、所望のノードを指定することにより、対象とされたノード間の優先順位を変更する。

【０１７１】図３２の超遂次グラフに対し、優先順位変更モードにおいて、プロセスＥとＦとの優先順位を変更するとする。ユーザは、上記のように入力装置４により、ノードＥ及びＦとを指定し、指定完了の旨を非決定性導入装置１７に与える。これにより、遂次化情報変更部３６は、遂次化情報記憶部３１内のプロセステーブルにおけるプロセスＥの優先順位フィールドＦ３の値を「２」、プロセスＦの優先順位フィールドＦ３の値を「１」と変更する。これにより、出力装置５上には、新たな超遂次グラフが表示されることとなる。図３５は、この時の超遂次グラフの一例を示す図である。図３５に示すように、優先順位変更後は、遂次構造に対するアークが変更されることになる。

【０１７２】このようなプロセスの優先順位の変更は、ユーザの意図したプロセス間の動作が保証されるため、良い非決定性部分の導入に有効に役立てることができる。すなわち、まず、ユーザは図３２の超遂次プログラムについてテスト実行装置１５で実行形式に変換し、テスト実行を行う。次に、ユーザは、優先順位変更後の図３５の超遂次グラフについて同様にテスト実行を行う。これにより、両方のテスト実行結果がユーザの意図したものであることが確認できれば、プロセスＥとプロセスＦは、どちらを先に実行しても良いとみなせるので、良い非決定性部分の導入を行うことができる。

【０１７３】なお、本実施例では、２プロセス間の非決定性の導入について説明したが、３プロセス以上の場合も同様である。

【０１７４】図３６は、３プロセス間の良い非決定性部分の導入を説明するための図である。並行構造のプロセ

スB、C、D間において、その全てについて並行化を許容するようにすることもできるが、図３６においては、プロセスBとCとの間のみ並行化を許容されている場合を示す。このような良い非決定性部分の導入により、超遅次グラフは図３７のようになる。つまり、図３７に示す超遅次グラフは、プロセスB及びCの両方が実行された後に、プロセスDが実行されることを示す。上記のように本実施例によれば、出力装置５上に表示した超遅次グラフによって並行化情報及び遅次化情報をユーザに対して同時に提示するので、ユーザはもとの並行構造を考慮しつつ、良い非決定性部分の指定をすることができるようになる。また、並行プログラム記述レベルにおける良い非決定性部分の指定・解除ではなく、超遅次グラフに対して良い非決定性部分を指定・解除をするので、高度な並行プログラミング技術を必要とすることなく、容易に並行プログラムの開発をすることができるようになる。

【０１７５】（実施例５）図３８は、本実施例に係る並行プログラム生成作成支援装置のブロック図であり、特に図２におけるＨＳＰファイル記憶部１４と非決定性導入装置１７について詳しく示す。

【０１７６】図３８において、ＣＰファイル記憶部１１に格納されている第１並行プログラムは、遅次化装置１２により先の実施例のように遅次化され、得られた超遅次プログラムはＨＳＰファイル記憶部１４内の遅次化プロセスリスト記憶部５１に格納される。なお、超遅次プログラムは、図３８には図示してないデバッグ装置により、遅次化プロセスリスト記憶部５１に格納される時には既に遅次化された状態でのバグが取り去られている。

【０１７７】ここで生成された超遅次プログラム（遅次化プロセス）は、フィールドデータ生成部５３によりフィールドデータに変換され、フィールドデータ記憶部５２に保存される。これら遅次化プロセスリスト記憶部５１とフィールドデータ記憶部５２の情報は、中間ファイルたるＨＳＰファイル記憶部１４に保存される。フィールドデータ記憶部５２からのフィールドデータは、フィールドチューニング部５４を介してフィールドエディタ５５での編集に供される。

【０１７８】非決定性導入装置１７は、フィールドデータ生成部５３、フィールドチューニング部５４及びフィールドエディタ５５で構成される。フィールドエディタ５５で編集が終了したフィールドデータは、並行化装置１８（フィールドデータ変換部）で修正された並行プログラムに変換され、第２ＣＰファイル記憶部１９に格納される。

【０１７９】図３９は、図３８におけるＣＰファイル記憶部１１に格納されている第１並行プログラムの一例を示しており、説明の便宜上、意図的にバグを含ませている。ここでは、バグとしてプロセスＰ４とＰ５の並行実行は誤りであり、逆に、プロセスＰ２とＰ６は並行実行が可能であるとし、正しい並行プログラムは後述する図６６であるとする。図４０は図３９の第１並行プログラムで実行されるプロセスの流れをイメージした図であり、プロセスＰ４とＰ５、プロセスＰ７とＰ８がそれぞれ並行に実行される。図４１は、図３８における遅次化プロセスリスト記憶部５１に格納されている遅次化プロセスの一例であり、遅次化装置１２において図３９の第１並行プログラムで並行な部分に、ある順番を付けることで遅次化されている。

【０１８０】また、図４２は図４１で実行されるプロセスの流れをイメージした図である。この例では偶然、Ｐ４、Ｐ５の順で遅次化されているので、この点に関するバグは取り除かれている。仮に、遅次化がＰ５、Ｐ４の順で行われていたなら、上記のバグが従来のデバック手法で検出されるはずであり、遅次化の時点で誤った並行化（悪い非決定性）を取り除くことができる。

【０１８１】図４３は、図３８におけるフィールドデータ生成部５３で実行される処理の流れを示すフローチャートである。ここでは、１つのプロセスからもう１つのプロセスへ処理が進むことをある制約と遷移条件が存在する範囲（エリア）ととらえて、超遅次プログラム（遅次化プロセス）をフィールドデータに変換している。図４３に示すように、ステップＤ１でスタートエリアを生成し、最後のプロセスが来るまでステップＤ３～Ｄ９を繰り返し実行する。そして、最後のプロセスが来たらステップＥ１０～Ｅ１２の処理を行って終了となる。図４４は、図４３に示した処理によって生成される一般的なエリアのデータ構造を示す。このエリアの接続で構成される全体構造としてフィールドが形成される。

【０１８２】エリアＡ（ｉ）には以下のような情報が記述されている。

（１）　エリアＡ（ｉ）にいる限りは、"constraint"に記述された制約が存在する。（例えば、Ｐ（ｘ１）はＰ（ｘ２）より先に実行される）

（２）　エリアＡ（ｉ）から他のエリアへ移るための遷移条件は、"transition"に記述されている。（例えば、Ｐ（ｚ１）が終了してＡ（ｊ）へ遷移できる）

（３）　エリアＡ（ｉ）から移ることのできるエリアは、Ａ(j) である。

（４）　エリアＡ（ｉ）へ移る以前の状態は、エリアＡ（ｋ）である。

【０１８３】図４５は、図４３に示した処理により図４１に記述されたプログラムをエリアの集合で構成されたフィールドに変換した様子を示す。

【０１８４】このフィールドは処理の順序関係の制約が最も強い。非決定性の導入は、この制約を変更したり、なくしたりすることで行なうことができる。つまり、フィールドチューニングを行うことで非決定性を導入し、並行化を図ることができる。なお、一般的には制約によるチューニングであるので、ユーザには制約と遷移条件

を表示する。

【０１８５】図４６は、図３８におけるフィールドチューニング部５４で実行される処理のフローチャートである。

【０１８６】ステップＥ１で選択されたフィールドデータは、ステップＥ２においてそのエリア間の連結情報が解析されて視覚的にフィールドが生成され、フィールドエディタ５５に表示される。ユーザは、フィールドエディタ５５の画面を見ながらステップＥ３でコマンドを入力し、フィールドのチューニングを行うことができる。

【０１８７】ステップＥ５で編集中のフィールド保存でき、編集終了ならステップＥ１７を経て終了する。なお、ステップＥ１４～Ｅ１６にて、編集による制約の矛盾を検出している。

【０１８８】ステップＥ９は制約の書き換え処理に関するサブルーチンである。また、ステップＥ１１は制約の追加処理に関するサブルーチンである。そして、ステップＥ１３は制約の削除処理に関するサブルーチンである。

【０１８９】図４７は図４６におけるステップＥ９の処理を行なうサブルーチンである。

【０１９０】ステップＥ９－１で編集された制約に関して、ステップＥ９－２で自明な制約の有無を検査し、もし自明な制約が存在すれば、制約の書き換えにエラーがあることをユーザにステップＥ９－９で伝え、ステップＥ９－１０で書き換えを無効にする。もし、自明な制約が存在しなければ、ステップＥ９－４のようにフィールドを書き換える。そして、ステップＥ９－５～Ｅ９－８でフィールドチェックする。

【０１９１】図４８は、図４６におけるステップＥ９－４の制約の変更操作のアルゴリズムによって起こるフィールドの変化を示した図である。これは"before Pi Pj"を"before Pl Pj"に変更したときの様子を示しており、図４８の（ａ）から（ｂ）へとフィールドが変化する。ここでは、Ａ（ｙ）のエリアがエリアＡ（ａ）の"next"に接続され、それに伴って各エリアの属性が変化する。

【０１９２】図４９は図４６におけるステップＥ１１の処理を行なうサブルーチンである。ステップＥ１１－１で編集された制約に関して、ステップＥ１１－２で自明な制約の有無を検査し、もし自明な制約が存在すれば、制約の書き換えにエラーがあることをユーザにステップＥ１１－９で伝え、ステップＥ１１－１０で追加を無効にする。もし、自明な制約が存在しなければ、ステップＥ１１－４のようにフィールドを書き換える。そして、ステップＥ１１－５～Ｅ１１－８でフィールドチェックする。

【０１９３】図５０は図４９におけるステップＥ１１－４の制約の追加操作のアルゴリズムによっておこるフィールドの変化を示した図である。ここではエリアＡ（ｙ）に"before Pj Pk"を追加したときの様子を示したもので、図５０（ａ）から（ｂ）へとフィールドが変化する。ここでは、Ａ（ｙ）のエリアが新たにエリアＡ（ｘ２）のnextに追加され、それに伴って各エリアの属性が変化する。

【０１９４】図５１は図４７におけるステップＥ１３の処理を行なうサブルーチンである。

【０１９５】エリアには少なくとも１つの制約が必要であるので、ステップＥ１３－１で指定したエリアＡ（ｙ）の制約の個数を調べ、もし１つならステップＥ１３－８でエラーメッセージを表示して終了する。もし、複数の制約があれば、ユーザの指定した制約を削除し、ステップＥ１３－３のようにフィールドを書き換える。そして、ステップＥ１３－４～Ｅ１１－７でフィールドチェックし、自明な制約の発生を防ぐ。

【０１９６】図５２は、図５１におけるステップＥ１３の制約の削除操作のアルゴリズムによって起こるフィールドの変化を示した図である。

【０１９７】これは"before Pj Pk"を削除したときの様子を示しており、図５２（ａ）から（ｂ）へとフィールドが変化する。ここでは、Ａ（ｘ２）のエリアがエリアＡ（ｚ）の"pre"に接続され、それに伴って各エリアの属性が変化する。

【０１９８】図５３はフィールドに矛盾を発生させる自明な制約を検出する処理であり、図４７のＥ９－２、Ｅ９－５、図４９のＥ１１－２、Ｅ１２－５、図５１のＥ１３－４で共通に実行されるサブルーチンである。ここでは、例えば"before A B"、"before B C"が存在するにも関わらず、"before A C"のような自明な制約を検出する。

【０１９９】図５３においてＥ９－３～Ｅ９－２－６の繰り返しにより、自明制約が生成され、Ｅ９－２－７で制約集合との間のに共通要素を探すことにより実現される。例えば、図５４のような自明制約"before A D"を含んだフィールドを仮定した場合、この処理によって図５５のテンポラリ集合（ａ）、テンポラリ集合（ｂ）、テンポラリ集合（ｃ）の順番で処理が実行され、共通要素"before A D"を検出することができる。

【０２００】図５６は、フィールドエディタ５５の表示画面を示す。フィールドエディタ５５では、フィールドの読み込み、保存、制約編集、表示モードの変更等をメニュー７１を指定して行うことができ、四角で表現されたエリア７２を選択すると、編集画面７３が現れる。図５７は、表示モードとして「２Ｄ表示モード」を選択した場合のフィールドの表示例を示す。このモードでは、平面的にフィールドが表示される。図５８は、表示モードとして「３Ｄ表示モード」を選択した場合のフィールドの表示例を示す。このモードでは、立体的にフィールドが表示され、全体が大きくなったり、多くの並行処理部分がある場合にも、全体を見渡しやすくすることができる。

【０２０１】図５９～図６６は、図４４に示したフィールドデータが図４６のアルゴリズムでチューニングされていく様子を示す。

【０２０２】図５９では、例えばユーザはＰ５とＰ６に因果関係がなく、Ｐ６はＰ２の後であれはバグとなる悪い非決定的な動きはしない判断したとする。この場合、図５９中に示されるように制約の書き換えるという編集を行うと、表示は図６０のようになる。図６０では、例えばＰ６とＰ４に因果関係はなく、Ｐ４はＰ３とのみ正しい順序が存在すると判断したとする。この場合、図６０中に示されるようにある制約（"before P6 P4"）を削除する編集を行うと、表示は図６１のようになる。以下同様に、ユーザがエリア内の制約を吟味し、徐々に編集を加えていくと、フィールドデータは図６２→図６３→…→図６６のように変化する。

【０２０３】図６７は、図３８における並行化装置（フィールドデータ変換部）１８で実行される処理を示すフローチャートである。図６７に示されるように、並行化装置１８ではフィールドデータ記憶部５２からフィールドデータを読み込み（ステップＦ１）、エリア間のグラフ構造を解析してプロセスフローを生成し（ステップＦ２）、更にそのプロセスフローを解析して非決定性のみ導入された形で並行プログラムのソースを生成し（ＳステップＦ３）、第２ＣＰファイル記憶部１９に保存する（ステップＦ４）。

【０２０４】図６８は、図６６の状態となったフィールドを並行化装置１８で解析して変換して得られた並行プログラムである。図６８に示されるように、図３９の第１並行プログラムで仮定したバグであるＰ４とＰ５の並行実行が直され、逆に、良い非決定性であるＰ３とＰ６の並行化が行われている。

【０２０５】また、図６９は図６８の並行プログラムで実行されるプロセスの流れをイメージした図である。

【０２０６】本実施例によれば、逐次化された並行プログラムのプロセスの流れを制約と遷移条件からなるフィールド（場）という概念として捉え、このフィールドをチューニングすることで良い非決定性を導入し、バグのない高品質な並行プログラムを効率的に生成することができる。また、本実施例では逐次化されたプログラムが制約を外すことによって並行化が進んでいくと、図形的に細長いフィールドだったものが縦方向に伸びていくので、直観的に理解しやすく、操作が容易となるという利点がある。

【０２０７】（実施例６）本実施例に係る並行プログラム作成方法及び作成支援装置では、テスト・デバッグは、超逐次プログラムに対して行う。また、良い非決定性に関する情報の導入も、超逐次プログラムに対して行う。

【０２０８】図７０は、実施例６に係る並行プログラム作成支援装置の概略構成を示す図である。図７０において、ＣＰファイル記憶部１１に格納された第１並行プログラムは、ユーザによる入力装置４からの指示により引き出され、逐次化装置１２に入力される。逐次化装置１２に入力された第１並行プログラムは、逐次化ルール記憶部１３に格納された逐次化ルールに従って完全超逐次プログラムＨＳＰに変換され、複数の完全超逐次プログラムを格納可能なＨＳＰファイル記憶部１４に記録される。

【０２０９】この完全超逐次プログラムＨＳＰに対して、テスト実行装置１５でテストが行われる。ここで、バグがあればデバッグ装置１６を用いて、ＨＳＰファイル記憶部１４に格納されている完全超逐次プログラムＨＳＰを修正する。テスト・デバッグが完了したならば、完全超逐次プログラムＨＳＰをＨＳＰファイル記憶部１４に保存する。

【０２１０】次に、非決定性導入装置１７で変更装置２５を介して完全超逐次プログラムＨＳＰの実行順序を変更し、別の完全超逐次プログラムＨＳＰを生成する。この完全超逐次プログラムＨＳＰに対して、同様のテスト・デバッグを繰り返しながら、完全超逐次プログラムＨＳＰをＨＳＰファイル記憶部１４に蓄積していく。すなわち、良い非決定性を１つの部分超逐次プログラムとして表現する替わりに、複数の完全超逐次プログラムとして表現したものである。非決定性導入装置１７による良い非決定性導入が完了したならば、最終的にＨＳＰファイル記憶部１４に蓄積された完全超逐次プログラムＨＳＰの集合から、並行化装置１８で並行プログラムが生成され、第２ＣＰファイル記憶部１９に格納される。

【０２１１】（実施例７）本実施例において並行プログラム作成支援装置の構成は、実施例６と同様であり（図７０）、その並行プログラム作成手順が異なっているので、図示は省略する。図７１は、本実施例における並行プログラム作成方法の概略手順を説明するためのフローチャートである。

【０２１２】（１）　ステップＡ１：モデル化
対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、各プロセス構造を決定する。更に、該プロセス内を並行プログラム等を用いたプログラミングにより、並行構造を有する並行プログラムをソースプログラムとして記述する。このソースプログラムには、バグが潜在的に存在する可能性がある。

（２）　ステップＡ２：逐次化
デフォルト逐次性を導入することによって、並行プログラムを逐次構造の超逐次プログラムに変換する。本実施例では、メタレベルにおいてデフォルト逐次性を導入する。ここで、メタレベルとはソースプログラムそのもののレベルについていうのではなく、ソースプログラムの実行を管理するレベルをいう。例えば、並行プログラムで記述されたソースプログラムを、それとは別に管理されるスケジューラによって逐次的に実行することを保障

したプログラムのソースプログラムに変換する。

【0213】（3） ステップA3：超遅次プログラムのテスト・デバッグ

ステップA2で変換された超遅次プログラムに対して、テスト・デバッグを行う。すなわち、テスト実行装置により遅次プログラムをテスト実行した結果に基づいて、デバッグ装置により超遅次プログラムからバグを除去する。ここでのテスト・デバッグは、遅次プログラムにおける通常のテスト・デバッグ方法と同様に行うことができる。超遅次プログラムが正常に動作することが保障されるまで、このテスト・デバッグを行う。

【0214】（4） ステップG1：並行模擬プログラム化

ステップA3でテスト・デバッグが行われた超遅次プログラムに対し、並行化の候補となるプロセス群を導入し、変更装置20によって複数の並行模擬動作系列を生成する。本実施例では、メタレベルにおいて並行模擬動作系列を導入する。

【0215】（5） ステップG2：並行模擬動作系列の実行・確認

ステップG1で得られた並行模擬動作系列をステップA3でテスト・デバッグが行われた超遅次プログラムを用いて実行し、実行結果の確認を行う。この実行は、プログラムが並行に動作する状況を超遅次プログラム上で模擬的に再現している。ユーザは、各動作結果が正しいか否かを並行模擬実行装置に記録する。

【0216】（6） ステップG3：良い非決定性の選択による部分遅次化

ステップG2の実行結果から、正しく実行された並行模擬動作系列を許容し、かつ実行の正しくない並行模擬動作系列を排除する非決定性を導入して、良い非決定性として登録する。ステップG1～ステップG3の処理を所定の回数繰り返し、良い非決定性の範囲を徐々に拡大していく。

【0217】（7） ステップA7：並行化コンパイル

良い非決定性の導入された部分超遅次プログラムのうち、無害な非決定性部分を抽出し、その部分を並行化することで、部分超遅次プログラム全体を並行プログラムに変換する。すなわち、並行化に関する情報が導入されなかった部分に関しては、デフォルト遅次性を並行プログラムに反映させて（例えば、ソースプログラム自体に埋め込む）、メタレベルのデフォルト遅次性を解除する。

【0218】図72は、ユーザが想定していた並行プログラムモデルの例である。ここで、P1、P2、P3等はプログラムが実行される単位を表しており、これらをプロセスと呼ぶ。同図の例は、ユーザの意図においてプロセスP3、P4、P5は並行実行され、プロセスP7、P8、P9は遅次実行されることを表している。更に、プロセスP2、P3、P4、P5、P6からなるプロセス群と、プロセスP7、P8、P9からなるプロセス群も並行実行可能である。

【0219】超遅次プログラムHSPは、遅次化装置12により並行モデルにおけるプロセスの前後関係を壊さない範囲で、全てのプロセスを一列に並べて得られる。

【0220】図73は、図72の並行プログラムから得られる超遅次プログラムHSPの実行系列をモデル化した例である。このように遅次化モデルでは、並行に実行されるプロセスP3、P4、P5等は任意の順で、場合によっては離れた位置に置かれるが、これらより先に実行されるべきプロセスP1、P2は、プロセスP3、P4、P5のいずれよりも前に置かれ、またこれらより後に実行されるべきプロセスP6、P10は、プロセスP3、P4、P5のいずれよりも後に置かれる。同様に、プロセスP2、P3、P4、P5、P6からなるプログラム群と、プロセスP7、P8、P9からなるプログラム群を遅次化した際の群の間の前後関係は任意で、場合によっては両方の群のプロセスが遅次モデル上に交互に表れることもある。

【0221】超遅次プログラムHSPは、ユーザによる入力装置4からの指示により、テスト実行装置15に入力され、テスト実行が行われる。テスト実行装置15は、テスト実行の結果を出力装置に提示する。ユーザは、このテスト実行の結果に基づいて超遅次プログラムHSPに対し、入力装置4を用いて所定のテスト・デバッグを行うことができる。また、ユーザは超遅次プログラムHSPに対して所定のテスト・デバッグを行った後、再度入力装置4からテスト実行の指示を与える。これにより、テスト・デバッグの行われた超遅次プログラムHSPは、テスト実行装置15に入力され、再度テスト実行が行われる。このテスト・デバッグは、プログラムHSPが正常に動作することを確認するまで繰り返し行われる。

【0222】テスト・デバッグにより正常に動作することを確認した時点で、非決定性入力装置17により超遅次プログラムを部分遅次プログラムへ変換する。まず、超遅次プログラムを複数の並行模擬プログラムへ変換し、並行模擬プログラムの実行結果より、良い非決定性に関する情報を蓄積する。この手順を図74に示す。この図74に示す手順に従い、図73の遅次モデルを並行模擬モデルへ変換し、部分遅次モデルを得るまでの過程の具体例を図75～図78に示す。

【0223】以下、この例の手順について説明する。

【0224】図75（a）ではポインティングデバイス等によって、並行模擬範囲としてP3、P4、P5が選ばれる（ステップH1）。更に、各々のプロセスが単独で並行化の単位に指定される（ステップH2）。図75では、並行化単位が色の違いによって区別される。超遅次プログラムでは、この並行模擬範囲の内部でプロセスはP3、P4、P5の順番で実行される。

【0225】図75（b）では、並行化単位の置き換えとしてP4、P5が置き換えられ（ステップH3）、一つの並行模擬動作系列が獲得される。この並行模擬動作系列では、並行模擬範囲の内部でプロセスはP3、P5、P4の順番で実行される。このプログラムを実行して実行結果が保証されると（ステップH4）、図75（c）のようなプログラムが獲得される（ステップH5）。この部分遂次プログラムでは、並行模擬範囲の内部でプロセスP3を実行した後にP4とP5が並行に実行される。

【0226】更に、同一範囲内での非決定性の拡大がユーザによって指示されると（ステップH6）、図76（a）では単独のプロセスP3と2つのプロセスの組P4&P5が置き換えられ（ステップH3）、もう一つの並行模擬動作系列が獲得される。この並行模擬動作系列では、並行模擬範囲の内部でプロセスはP4とP5が並行に実行された後にP3が実行される。このプログラムを実行して正しさが確認されると（ステップH4）、図76（b）のような非決定性を持つプログラムが獲得される（ステップH5）。この部分遂次プログラムでは、並行模擬範囲の内部でプロセスP3、P4、P5が並行に実行される。これによって、この並行模擬範囲内での非決定性は十分となるので、ステップH6からステップH7に進み、並行範囲の拡大がユーザによって指示されると（ステップH7）、図77～図78に示すように、ステップH1に戻って同様の処理を繰り返す。

【0227】良い非決定性に関する情報がインクリメンタルに導入され得られた部分遂次プログラムPHSPは、ユーザからの指示により、並行化装置18に入力される。この並行化装置18では、良い非決定性のみを持つ部分遂次プログラムPHSPを並行プログラムCPとして第2CPファイル記憶部19に記録する。ユーザは、このプログラムCPを出力装置5によって見ることができるとともに、最終的なテスト・デバッグを行うことができる。

【0228】本実施例によれば、超遂次プログラム上で並行プログラムの動作を十分に確認することができる。また、超遂次プログラムに対して並行化の候補となる範囲を指定することで、超遂次プログラムを段階的に並行プログラムへ変換することができる。更に、並行模擬動作系列に基づく遂次実行で正しく動作することが確認された非決定性のみを許容する並行性を導入することで、正しく動作する並行プログラムを得ることができる。これによって、並行プログラムのテスト・デバッグが更に容易となるという利点がある。

【0229】（実施例8）本実施例は、図1のように構成されたコンピュータシステムにおいて、図79に示すように各プロセッサ1－1～1－Nに少なくとも1個以上のプロセスA～Dが登録され、それぞれのプロセスは並行／並列に動作しており、互いにメッセージを交換しながら情報交換して処理を行うというプロセス実行環境において、並行プログラムの作成を行う例である。

【0230】本実施例では、並行プログラムのテストを超遂次プログラムに対して行うが、バグのあった場合はソースコードたる並行プログラムを修正せずに、遂次化ルールを修正することでテスト・デバッグを行う。

【0231】図80は、本実施例に係る並行プログラム作成支援装置の概略構成を示す図である。

【0232】図80において、CPファイル記憶部11に格納されている第1並行プログラムは、遂次化装置12において遂次化ルール記憶部13に格納された遂次化ルールに従って、メタレベルにおいてデフォルト遂次性が導入されることにより、超遂次プログラム（HSP）に変換され、HSPファイル記憶部14に記録される。ここで、遂次化ルールはソースプログラム（CPファイル）を実行装置22で実行させた際に得られる実行ログ情報となり、HSPファイル記憶部14はCPファイルと遂次化ルールである実行ログ情報の対からなる。ユーザは、この超遂次プログラムHSPを出力装置5によって見ることができる。この超遂次プログラムHSPは、ユーザによる入力装置4からの指示によってテスト実行装置15に入力され、テスト実行が行われる。

【0233】テスト実行装置15は、テスト実行の結果（実行ログ）を出力装置5に提示する。ユーザは、このテスト実行の結果に基づいて、遂次化ルール修正装置26を用いて遂次化ルール記憶部13に格納されている遂次化ルールを修正し、超遂次プログラムHSPに対し所定のテスト・デバッグを行うことができる。

【0234】ユーザは、このように遂次化ルールの修正により超遂次プログラムHSPに対して所定のテスト・デバッグを行った後、入力装置4から再度テスト実行の指示を与える。これにより、テスト・デバッグの行われた超遂次プログラムHSPは、テスト実行装置15に入力され、再度テスト実行が行われる。このテスト・デバッグは、超遂次プログラムHSPが正常に動作することを確認するまで繰り返し行われる。

【0235】そして、本実施例ではテスト・デバッグにより正常に動作することを確認した時点で、遂次化ルール修正装置26により遂次化ルール記憶部13に格納されている遂次化ルールが修正され、この修正の後の遂次化ルールが遂次化ルール記憶部13に新たに記録される。この遂次化ルールの修正によって、超遂次プログラムに対して良い非決定性に関する情報が部分的に導入される。超遂次プログラムHSPは部分超遂次プログラムPHSPに変換される。

【0236】次に、部分超遂次プログラムPHSPは、潜在的に存在する無害な並行性を顕在化させるために、並行化装置18にて部分超遂次プログラムPHSPにおける遂次化ルールを分割基準指定装置23からの指示に基づいて各プロセス毎に分割し、各プロセス毎の起動順

序制御を可能とする第２並行プログラム１９に変換する。

【０２３７】図８１は、本実施例に係る並行プログラム作成方法の概略手順を説明するための図である。

【０２３８】（１）　ステップＡ１：モデル化

対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、各プロセス構造を決定する。更に、該プロセス内を並行プログラム等を用いたプログラミングにより、並行構造を有する第１並行プログラムをソースプログラムとして記述する。このソースプログラムには、バグが潜在的に存在する可能性がある。

【０２３９】（２）　ステップＡ２：逐次化

デフォルト逐次性を導入することによって、第１並行プログラムを逐次構造の超逐次プログラムに変換する。本実施例では、メタレベルにおいてデフォルト逐次性を導入する。ここで、メタレベルとはソースプログラムそのもののレベルについていうのではなく、ソースプログラムの実行を管理するレベルをいう。例えば、並行プログラムで記述された第１並行プログラムを、それとは別に管理されるスケジューラによって逐次的に実行することをいう。すなわち、ここでいう逐次化とは、ＣＰファイル記憶部１１と逐次化ルール記憶部１３を対応付けることである。よって、得られる超逐次プログラムは、第１並行プログラムと逐次化ルールの対からなるとする。

【０２４０】（３）　ステップＡ３：超逐次プログラムのテスト・デバッグ

ステップＡ２で逐次化された超逐次プログラムを実行し、テストを行う。この際のテスト実行は、第１並行プログラムを逐次化ルールに基づいて動作させることをいう。機能面のバグがある場合は、第１並行プログラムを修正し、ステップＡ２へ戻る。タイミング面のバグがある場合は、ステップＪ１に移行する。タイミング面のバグがない場合は、その逐次化ルールを保存してステップＡ４に移行する。

【０２４１】（４）　ステップＪ１：逐次化ルールの修正

ステップＡ３でタイミング面のバグがある場合は、逐次化ルールを修正した後、ステップＡ２に戻る。

【０２４２】（５）　ステップＡ４：良い非決定性の導入

良い非決定性の追加が必要な場合は、逐次化ルールを修正した後、ステップＡ２に戻る。良い非決定性の追加が必要ない場合は、ステップＡ７に移行する。

【０２４３】（６）　ステップＡ７：並行化コンパイル

ステップＡ３で保存された逐次化ルールから無害な非決定性部分を抽出し、その部分を並行化することで、部分超逐次プログラム全体を第２並行プログラムに変換する。

【０２４４】図８２は、図７９に示したようなプロセス間通信のある並行プログラムをメタレベルにおいてデフォルト逐次性を導入して実行した結果の実行履歴であるログ情報の一例を示す図である。

【０２４５】ユーザは、このログ情報を出力装置５によって見ることができる。図８２において、縦のスクロールバーは任意の時刻のログ情報を見るために表示情報をスクロールするために用い、横のスクロールバーは良い非決定性を導入し複数の実行可能な候補がある時に、それら情報を見るために用いる。このように縦と横のスクロールバーを用いることにより、全ログ情報中の任意の場所を見ることができる。更に、画面上部に示された「順序修正」と「非決定性導入」は、後述するそれぞれの操作を要求する時に、選択して処理を指示するために用いる。

【０２４６】更に、ユーザは入力装置４を介して指示を行うことによって、ＨＳＰファイル記憶部１４に保存されている第１並行プログラムと実行ログ情報ファイルをテスト実行装置１５に入力し、超逐次プログラムＨＳＰとしてテスト実行を行うことができる。

【０２４７】テスト実行装置１５は、テスト実行の結果を出力装置５に提示する。ユーザは、このテスト実行の結果に基づいて、超逐次プログラムＨＳＰに対し入力装置４を用いて所定のテスト・デバッグを行うことができる。テスト・デバッグは、機能面の不具合についてはＣＰファイル記憶部１１内の第１並行プログラムを修正し、タイミング面の不具合については逐次化ルール記憶部１３内の実行ログ情報ファイルを修正し、再度、修正された第１並行プログラムと実行ログ情報を対応付けし直し、超逐次プログラムを再作成する。

【０２４８】ユーザは、超逐次プログラムＨＳＰに対して前記テスト・デバッグを行った後、再度入力装置４からテスト実行の指示を与える。これにより、テスト・デバッグの行われた超逐次プログラムＨＳＰは、テスト実行装置１５に入力され、再度テスト実行が行われる。このテスト・デバッグは、超逐次プログラムＨＳＰが正常に動作することを確認するまで繰り返し行われる。

【０２４９】テスト・デバッグにより正常に動作することを確認した時点で、逐次化ルール修正装置２６により超逐次プログラムに対して良い非決定性に関する情報を部分的に導入していく。逐次化ルール修正装置２６より導入された良い非決定性に関する情報は、ログ情報に反映され、ＨＳＰファイル記憶部１４に記録される。

【０２５０】次に、逐次化ルール修正装置２６により良い非決定性に関する情報が導入された超逐次プログラム（部分超逐次プログラムＰＨＳＰ）について、ユーザからの指示に従ってテスト実行装置１５によりテスト実行を行い、テスト・デバッグを行う。この場合、部分超逐次プログラムＰＨＳＰの振る舞いは、良い非決定性に関する情報の導入された部分については、非決定的な振る舞いをするので、その振る舞い全てについてテスト・デバッグを行うことが好ましい。このようにして、テスト

・デバッグ及び良い非決定性に関する情報の導入を繰り返し、良い非決定性に関する情報を徐々に付加していく。

【０２５１】良い非決定性に関する情報がインクリメンタルに導入され得られた部分超遅次プログラムＰＨＳＰは、ユーザからの指示により並行化装置１８に入力される。この並行化装置１８では、部分超遅次プログラムＰＨＳＰのうち、無害な非決定性部分を抽出し、部分超遅次プログラムＰＨＳＰ全体を並行化する。即ち、集中ログ情報を各プロセス毎にその起動順序を規定するように分割し、各プロセス毎に起動制御を行うようにすることで、デフォルト遅次性を解除し、部分超遅次プログラムを並行プログラムに変換して第２ＣＰファイル記憶部１９に記録する。ユーザは、この並行プログラムを出力装置によって見ることができるとともに、最終的なテスト・デバッグを行うことができる。

【０２５２】図８３は、まず、ステップＡ２～ステップＡ４の処理に関する部分超遅次プログラムを構成するまでの部分の構成例を示す。以下に、それぞれの構成部分の機能実現法について説明する。

【０２５３】図８４に、プロセス間で交換されるメッセージのフォーマットを示す。「宛先情報」には宛先プロセス名等の情報伝達先を指定する情報がセットされ、「送信元情報」には送信元プロセス名等の送信元を規定する情報がセットされる。「メッセージ本体」には、プロセス間で交換するデータがセットされる。

【０２５４】ログ情報として保存される情報は、図８４のメッセージフォーマット中の宛先情報と送信元情報を用いて生成される。以下の説明においては、ログ情報として宛先情報として宛先プロセス名を、送信元情報として送信元プロセス名だけを用いるが、同一プロセスが複数回利用される場合は、何回目の使用かの判別がつかなくなるので、送信元情報を｛プロセス名とプロセス起動回数｝からなるように拡張することができる。更に、あるプロセスの１回の起動中に複数回のメッセージ送信がある場合は、今回の起動後の何回目の送信かを特定するため、送信元情報を｛プロセス名、プロセス起動回数、メッセージ送信回数｝からなるように拡張することができる。更に、各プロセスが複数の処理の単位（メソッドと以降呼ぶ）を持つ場合には、その情報も付加して送信元情報は｛プロセス名、メソッド名、メソッド起動回数、メッセージ送信回数｝とする事もできる。

【０２５５】また、宛先情報も宛先プロセス名だけでなく、宛先メソッド名を付加して｛宛先プロセス名とメソッド名｝とすることもできる。もちろん、実行形態に応じて宛先情報と送信元情報を組み合わせることができるため、計８通りの組み合わせが存在する。

【０２５６】まず、実行装置２２について詳述する。前述したように、第１並行プログラムをランダムに実行する。その際の実行ログを遅次化ルールとするため、本実施例では、遅次化ルールとしてプロセス間の全ての通信を一旦プロセス群制御部２０３に送り、処理待ちメッセージ保存部２０４に蓄える。プロセス群制御部２０３は、処理待ちメッセージ保存部２０４から順次メッセージを取り出し、本来の宛先に対して再度送る。よってプロセス群制御部２０３によって全プロセスの起動が遅次的に制御されるため、第１並行プログラムは遅次的に処理されるようになる。

【０２５７】この時、プロセス群制御部２０３によって遅次的に処理されたメッセージは、ログ情報取得部２０２によって時系列にログ情報としてログ情報記憶部２０１に保存される。このログ情報が遅次化ルール記憶部１３となり、以降の遅次実行時の実行制約条件となる。

【０２５８】図８５は、プロセス間の通信をプロセス群制御部２０３を経由して行う様子を示す。この例では、プロセスＡとＢがプロセスＣにメッセージを送信する時、プロセスＡとＢのどちらのメッセージが先にプロセスＣに届いても良いとする良い非決定性が存在する場合であるが、実行時にたまたまプロセスＡがプロセスＢより先にメッセージを送信したとすると、プロセス群制御部２０３において｛プロセスＡ→プロセスＣ｝が先にログ情報に記憶され、続いて｛プロセスＢ→プロセスＣ｝が記憶される。図８６に、その場合の遅次化ルールであるログ情報の一例を示す。

【０２５９】次に、遅次化装置１２において、ＣＰファイル記憶部１１に記憶されている第１並行プログラムと上記のようにして得られた遅次化ルールであるログ情報を対応付ける。これにより、並行性を持った第１並行プログラムは、遅次化情報が付加された超遅次プログラムに変換される。

【０２６０】次に、テスト実行装置１５におけるテスト・デバッグの段階では、プロセス間で交換されるメッセージをプロセス群制御部２０３において一旦受信し、ログ情報記憶部２０１に記憶されたログ情報の順序に従って本来の宛先に対して送信するようにする。このためプロセス群制御部２０３は、処理待ちメッセージ保存部２０４に保存されたメッセージ中から、ログ情報に従った順序でメッセージを取り出して送信する機構を有している。

【０２６１】上記のように、プロセス群制御部２０３でメッセージ送信順序をログ情報に指定された順序で送信するようにすることにより、並行プログラムの難しさの一つである処理順序の非決定性を解決することができ、処理の再現性が実現される。つまり、図８５の例において、あるテスト実行時にはプロセスＢの方がプロセスＡより先にメッセージをプロセスＣに送ったとしても、プロセス群制御部２０３でログ情報に図８６に示すようにプロセスＡからのメッセージを先にプロセスＣに送る旨が記載されている場合は、プロセスＢからのメッセージを処理待ちメッセージ保存部２０４に保存しておき、プ

ロセスＡからのメッセージを受信しプロセスＣに対して送信した後、処理待ちメッセージ保存部２０４に保存しておいたプロセスＢからのメッセージをプロセスＣに送る。

【０２６２】図８７に、プロセス群制御部２０３の処理の流れを示す。

【０２６３】ログ情報記憶部２０１に保存されたログ情報に従って実行順番が一意に規定された超遅次プログラムにおけるテスト・デバッグは、機能面に不具合がある場合はソースプログラムを修正して上記処理ステップのステップＡ２から再度行う必要があるが、タイミングのみに不具合がある場合は、容易にテスト・デバッグをする事ができる。つまり遅次化ルールであるログ情報を修正し意図する順番にログ情報を書き換えれば良い。この書換は、ログ情報修正部２０５によって行う。遅次化ルールであるログ情報の修正は、ユーザからの指示により遅次化ルール修正装置２６内のログ情報修正部２０５において行う。ログ情報修正部２０５は、図８２に示すようにその処理した順番（時系列）にログ情報を表示することができる表示部を持つ。そしてタイミングのバグを修正したいユーザは、ログ情報の順序を入れ換える入替部を起動し、図８８に示すように処理順序を交換したいメッセージを指定する。図８８の例では、２メッセージ間の順序入れ替えを指示したが、複数メッセージの順序を入れ換えることも可能である。もちろん１メッセージ以上のメッセージを１つの集合としてとらえ、集合間での交換も可能である。そして、ログ情報修正部２０５には、ログ情報記憶部２０１のログ情報を入替部によって指定されたように書き換える書換部があり、図８８の例の場合は図８９のように変更される。また入れ替えメッセージの指定法としては、ポインティングデバイスを使って該当メッセージを指定する方法や、画面上の行番号をキーボードから指定する方法等がある。

【０２６４】上記のように、ログ情報を意図する順序に並べ替えることによって、テスト実行時にプロセス群制御部２０３が受信メッセージを本来の宛先プロセスに送信する順番を変えることになるため、タイミングの不具合を容易に修正することができるようになる。

【０２６５】例えば、図８５において本来プロセスＣはプロセスＢからのメッセージをプロセスＡからのメッセージより先に処理しなければならない場合に、ログ情報に図８６に示すようにプロセスＡからのメッセージを先に処理するようになっていたならば、上記の手順に従ってログ情報を図８９に示すように修正すればよい。これによって、プロセス群制御部２０３のメッセージ処理順番が変更されるため、処理のタイミングの不具合を解決することができる。

【０２６６】このようなタイミングの不具合解決方法を用いることによって、プロセスの処理順序を変更するためにソースプログラムを修正することなく、容易にテスト・デバッグをすることが可能となり、プログラム開発の生産性が向上する。

【０２６７】次に、良い非決定性の導入方法について説明する。

【０２６８】良い非決定性は、ログ情報修正部２０５内の非決定性導入部によってログ情報を修正することで行う。これは、超遅次化されたプログラムにユーザが意図的に非決定性を導入するものである。例えば図８５に示す処理において、プロセスＡとプロセスＢのどちらが先にプロセスＣにメッセージを送っても良い場合には、対象となるログ情報を図９０に示すように指定することによって行う。これによって、非決定性導入部は、指定されたメッセージの順序制約を解除した形で、例えば図９１（ａ）、図９１（ｂ）に示すようにログ情報記憶部２０１に記憶されたログ情報を修正する。図９１（ｂ）は、図９１（ａ）を横スクロールバーで右へシフトするように指示することによって得られる。この場合は２つのメッセージ順序の非決定性を導入した場合であるが、更に多数のメッセージ順序の非決定性を導入する場合は、順次右へシフトすることによってその一覧を見ることができる。

【０２６９】図９１では、プロセスＡないしプロセスＢからのメッセージであれば、プロセスＣにメッセージを送り、続いて次のタイミングでも、プロセスＡないしプロセスＢからのメッセージであれば、プロセスＣにメッセージを送るように解釈する。プロセス群制御部２０３が上記のようにログ情報を解釈するようにすることによって良い非決定性を扱うことが可能となる。これは、プロセス群制御部２０３の処理における図８７におけるステップＫ３において、メッセージ送信元として一致するか否かのチェック対象が、一つから複数個に拡張されることによって実現できる。

【０２７０】但し、上記例では、プロセスＣがプロセスＡないしプロセスＢから続けて２つのメッセージを受信した場合でも、それらのメッセージを処理してしまう問題点があるが、これは、先に受信したメッセージを次の起動候補から削除することによって容易に防止することができる。削除対象となるメッセージは、本実施例では説明の容易のために、送信先情報をプロセス名のみで示したが、図８４の送信先情報説明で前述したように唯一に特定できるので、未処理メッセージの情報を削除することはない。

【０２７１】よって、この良い非決定性の導入によって、外部からの非決定的な刺激に適切に反応でき、プロセスの柔軟性、再利用性、拡張性を実現できる。

【０２７２】上記の実施例では、良い非決定性を２つのメッセージの到着順序に関して示したが、２つ以上の複数のメッセージの到着順序に良い非決定性を与えることも可能である。その場合でも、上記処理と同様にプロセス群制御部２０３の処理における図８７のステップＫ３

においてメッセージ送信元として一致するか否かのチェック対象を、一つから複数個に拡張することによって解決されるので、正しく動作することは明らかである。

【０２７３】次に図８１におけるステップＡ７と図８０の並行化装置１８に関する並行化コンパイルのフェーズについて詳述する。図９２に、ステップＡ７の処理に関する部分の構成例を示す。ここで、集中ログ情報記憶部３０１はＨＳＰファイル記憶部１４内に保存されている良い非決定性導入後の実行ログ情報を保存している記憶部である。例えば図７９におけるプロセス群が図９３に示すようなメッセージ交換を行った場合、図８２で示したような超逐次プログラム用の集中ログ情報から、例えば図９４のように良い非決定性が導入された集中ログ情報に変換されたものが保存されているとする。

【０２７４】以下には、良い非決定性が導入された集中ログ情報を各プロセス毎に分割する手順を図９２、図９４、図９５を用いて説明する。集中ログ情報を分割する基準は、分割基準指定装置２３内の分割基準指定部３０３から指定する。この基準としては、例えば宛先プロセス名であったり、宛先計算機番号であったりするが、以下の説明では、宛先プロセス名を用いる。宛先名にメソッド名も付加する場合には、分割基準として、｛宛先プロセス名、メソッド名｝を指定して、宛先メソッド毎に分割して分割ログを作ることも可能である。分割基準指定部３０３では、この分割基準を保持している。

【０２７５】ユーザからの指示により集中ログ情報を分割させようとすると、ログ情報分割部３０２が起動される。ログ情報分割部３０２は、図９５に示す手順に従って集中ログ情報を分割基準指定部３０３によって指定された基準で分割する。

【０２７６】ログ情報分割部３０２が起動されると、分割基準指定部３０３より分割基準値を読み込み（ステップＬ１）、分割基準を把握する。続いて集中ログ情報記憶部３０１に保存されているログ情報を１レコード読み込み（ステップＬ２）、最終レコードか否かチェックする（ステップＬ３）。この例では、集中ログ情報の１レコード目は、図９４に示すようにプロセスＡからプロセスＤへのメッセージなので、最終レコードでないと判断し、分割基準（宛先プロセス名＝プロセスＤ）に従って、該当する分割ログ情報記憶部（ここでは、分割ログ情報記憶部Ｄとする）に保存する（ステップＬ４）。この操作を集中ログ情報記憶部３０１に記憶されているログ情報が終了するまで繰り返す。この結果、図９６に示すようにあて先プロセス毎に分割されたログ情報が分割ログ情報記憶部Ａ、Ｂ、Ｃ、Ｄのそれぞれに保存される。

【０２７７】図９６（ａ）では、例えばプロセスＡはプロセスＣからのメッセージを処理した後に、プロセスＤからのメッセージを処理することを規定している。また、図９６（ｃ）のプロセスＣは、いわゆる良い非決定性を導入した処理であり、プロセスＡないしプロセスＢからのメッセージを最初に処理し、次もプロセスＡないしプロセスＢからのメッセージを処理することを規定している。これは言い替えると、プロセスＡからのメッセージとプロセスＢからのメッセージのどちらが先に来ても良いという良い非決定性を表現している。ここでは、２つのメッセージの順序の制約を解除した例を示したが、より多くのメッセージの順序制約を解除し、良い非決定性を導入したい場合は、同一行に所望数のメッセージを列挙すれば良い。

【０２７８】また、分割ログ情報は宛先プロセス毎に分割されるため、分割ログ情報中の送信先プロセス名を省略することもできる。

【０２７９】次に、このようにプロセス毎に分割されたログ情報に従って各プロセスが動作する方式の一例を述べる。各プロセスは、例えば図９７に示すようにメッセージを受信し実際のプロセスの処理を開始する前にプロセス制御部３０５を呼出し、図９８に示す処理を行う。このようなプロセス制御部の呼出は、並行化装置１８の一部として第１並行プログラムにおける各プロセスの処理の先頭に呼出処理を挿入するようにソースプログラムを書き換えることによって行うことができる。即ち、各プロセスは図９９に示すように、当該プロセス本来の処理部分に加え、プロセス制御部３０５、分割ログ情報記憶部３０４、処理待ちメッセージ保存部３０６からなる。プロセス制御部３０５は、当該プロセスに内在する分割ログ情報記憶部３０４に保存されているログ情報を参照してメッセージ処理を行い、処理すべきメッセージでない場合は、処理待ちメッセージ保存部３０６に保存する。

【０２８０】次に、プロセス制御部３０５の処理について図９６と図９８を用いて説明する。

【０２８１】例えばプロセスＡは、図９３に示す処理の流れの場合、プロセスＣとプロセスＤのどちらから先にメッセージをもらうかは決定されておらず、ここに処理タイミングの非決定性が存在している。しかし、ここでは超逐次で実行したときの実行ログ（図８２相当）より、プロセスＣからのメッセージを先に処理し、その後にプロセスＤからのメッセージを処理するように規定される。よって、無害の非決定性を導入したとしても、このメッセージ処理順序の規定は守られなければならない。

【０２８２】最初の例として、プロセスＡが最初にプロセスＣからのメッセージを受信して起動された場合を示す。プロセスＡがプロセスＣからのメッセージを受信して起動されたならば、プロセス制御部が呼び出され、当該プロセスを起動させたメッセージの送信元情報を得る（ステップＭ１）。ここでは、送信元情報Ｉｓ＝｛プロセスＣ｝となる。続いて分割ログ情報記憶部よりまず最初に来るべきメッセージ情報Ｉｒ１を得る（ステップＭ

２）。ここでは、Ｉｒ１＝｛プロセスＣ｝となる。

【０２８３】次に、現在受信したメッセージが分割ログ情報に記憶された順序に一致しているかを調べるため、ＩｓがＩｒ１に含まれているかどうかを調べる（ステップＭ３）。この場合は、ともにプロセスＣで一致し含まれているため、受信メッセージに基づいた処理を即行うことが許される。よって次の起動チェックのために分割ログ情報記憶部から読み出すレコードを１つ進めておく（ステップＭ４）。そして、プロセス本体の処理を開始する前に、処理待ちメッセージ保存部に実行待ちメッセージが存在している場合には、その保存されたメッセージが起動可能となったかどうかを調べる。つまり、分割ログ情報記憶部に保存されている情報から次に処理すべきメッセージ情報Ｉｒ２を取り出し（ステップＭ５）、それに含まれるメッセージが処理待ちメッセージ保存部に存在しているか調べる（ステップＭ６）。この場合は、Ｉｒ２＝｛プロセスＤ｝となるが、処理待ちメッセージ保存部にメッセージが存在しないため処理を終了し、プロセスＣからのメッセージに基づいた本処理を行う。

【０２８４】続いて、プロセスＡ宛にプロセスＤからのメッセージが届いたならば、上記同様に処理を行うと、Ｉｓ＝｛プロセスＤ｝、Ｉｒ１＝｛プロセスＤ｝となり、集合Ｉｓは集合Ｉｒ１と等しいか又はＩｒ１に含まれる。Ｉｒ２は存在しないため、処理待ちメッセージ保存部内のメッセージを調べる必要なく処理を終了し、プロセスＤからのメッセージに基づいた処理を行う。

【０２８５】また、逆にプロセスＡが、プロセスＤから先にメッセージを受信した場合の処理の流れを示す。つまりプロセスＡが、最初にプロセスＤからのメッセージを受信して起動されたならば、プロセス制御部が呼び出され、当該プロセスを起動させたメッセージの送信元情報を得る（ステップＭ１）。ここでは、送信元情報Ｉｓ＝｛プロセスＤ｝となる。続いて、分割ログ情報記憶部よりまず最初に来るべきメッセージ情報Ｉｒ１を得る（ステップＭ２）。ここではＩｒ１＝｛プロセスＣ｝となる。

【０２８６】次に、現在受信したメッセージが分割ログ情報に記憶された順序に一致しているかどうかを調べるため、ＩｓがＩｒ１に含まれているか調べる（ステップＭ３）。この場合は、含まれないため即プロセスＤからのメッセージを処理してはいけないと判断し、処理待ちメッセージ保存部３０６にプロセスＤからのメッセージを保存し（ステップＭ８）、プロセスＡの処理を強制的に終了させる（ステップＭ９）。即ち、プロセスＤからのメッセージに基づいた処理を行うことなくプロセスＡの処理を終了する。

【０２８７】続いて、プロセスＡ宛にプロセスＣからのメッセージが届いたならば、上記同様に処理を行うと、Ｉｓ＝｛プロセスＣ｝、Ｉｒ１＝｛プロセスＣ｝となりＩｓはＩｒ１に含まれるため、受信メッセージに基づいた処理を即行うことが許される。よって次の起動チェックのために分割ログ情報記憶部から読み出すレコードを１つ進める（ステップＭ４）。そして、プロセス本体の処理を開始する前に、処理待ちメッセージ保存部に起動可能となった実行待ちメッセージが存在しているかどうかを調べる。つまり、分割ログ情報記憶部に保存されている情報から次に処理すべきメッセージ情報Ｉｒ２を取り出し（ステップＭ５）、それに含まれるメッセージが処理待ちメッセージ保存部に存在しているか調べる（ステップＭ６）。この場合は、Ｉｒ２＝｛プロセスＤ｝となり、処理待ちメッセージ保存部に先ほど保存しておいた該当メッセージが存在するため、その実行可能となったメッセージを次に処理するために、送信先（ここではプロセスＡ）宛に送信元情報等をそのまま（この例では、プロセスＤのまま）送信する（ステップＭ７）。そして、プロセスＡは受信メッセージとしてプロセスＣから送られたメッセージに基づく処理を行うべくプロセス制御部の処理を終了する。

【０２８８】更に、プロセスＡは前記処理待ちメッセージ保存部から取り出し送信したメッセージを受信するので、上記同様に処理を行うと、Ｉｓ＝｛プロセスＤ｝、Ｉｒ１＝｛プロセスＤ｝となりＩｓはＩｒ１となる。Ｉｒ２はもはや存在しないため、処理待ちメッセージ保存部内のメッセージを調べる必要なく処理を終了し、プロセスＤからのメッセージに基づいた処理を行う。

【０２８９】このことより、たとえメッセージの到着順序が実行ログに記載された順序と異なっていたとしても、実行ログに記載された順序に処理され、非決定的な処理の再現性が保証される。

【０２９０】次に良い非決定性を導入した場合の処理例として、プロセスＣの動作を例に示す。プロセスＣが最初にプロセスＡからのメッセージを受信して起動されたならば、プロセス制御部が呼び出され、当該プロセスを起動させたメッセージの送信元情報を得る（ステップＭ１）。ここでは、送信元情報Ｉｓ＝｛プロセスＡ｝となる。続いて分割ログ情報記憶部よりまず最初に来るべきメッセージ情報Ｉｒ１を得る（ステップＭ２）。ここではＩｒ１＝｛プロセスＡ、プロセスＢ｝となり、プロセスＡからのメッセージでもプロセスＢからのメッセージでも、どちらからのメッセージでも良いとしている。

【０２９１】次に、現在受信したメッセージが分割ログ情報に記憶された順序に一致しているかを調べるため、ＩｓがＩｒ１に含まれているかどうかを調べる（ステップＭ３）。この場合、集合Ｉｓは集合Ｉｒ１と等しいか又はＩｒ１に含まれるため、受信メッセージに基づいた処理を即行うことが許される。よって、次の起動チェックのために分割ログ情報記憶部から読み出すレコードを１つ進めておく（ステップＭ４）。そして、プロセス本体の処理を開始する前に、処理待ちメッセージ保存部に

実行待ちメッセージが存在している場合には、その保存されたメッセージが起動可能となったかを調べる。つまり、分割ログ情報記憶部に保存されている情報から次に処理すべきメッセージ情報Ｉｒ２を取り出し（ステップＭ５）、それに含まれるメッセージが処理待ちメッセージ保存部に存在しているか調べる（ステップＭ６）。この場合は、Ｉｒ２＝｛プロセスＡ、プロセスＢ｝となるが、処理待ちメッセージ保存部にメッセージが存在しないため、処理を終了し、プロセスＡからのメッセージに基づいた本処理を行う。

【０２９２】続いて、プロセスＣ宛にプロセスＢからのメッセージが届いたならば、上記同様に処理を行うと、Ｉｓ＝｛プロセスＢ｝、Ｉｒ１＝｛プロセスＡ、プロセスＢ｝となり、ＩｓはＩｒ１と等しいか又はＩｒ１に含まれる。Ｉｒ２は存在しないため、処理待ちメッセージ保存部内のメッセージを調べる必要なく処理を終了し、プロセスＢからのメッセージに基づいた処理を行う。

【０２９３】上記の良い非決定性導入の例では、メッセージの到着順序が逆の状態、即ちプロセスＣに、プロセスＢからのメッセージが先に到着し、その後にプロセスＡからのメッセージが到着した場合も同様に処理できることは明かである。

【０２９４】但し、上記例では、プロセスＣがプロセスＡないしプロセスＢから続けて２つのメッセージを受信した場合でも、それらのメッセージを処理してしまう問題点があるが、これは、先に受信したメッセージを次の起動候補から削除することによって容易に防止することができる。削除対象となるメッセージは、本実施例では説明の容易のために、送信先情報をプロセス名のみで示したが、図８４の送信先情報説明で前述したように唯一に特定できるので、、未処理メッセージの情報を削除することはない。

【０２９５】上記のように、オブジェクト毎に実行ログ情報を分割して動作させることによって、並行プログラムが潜在的に持っている無害な非決定性を実現しながら、超遅次プログラムで抽出した遅次性を維持しているため、処理の再現性を保持しており、超遅次プログラム実行時と同一結果を、より高い処理性能で実現することが可能となる。

【０２９６】また、上記の例では良い非決定性を２つのメッセージの到着順序に関して示したが、２つ以上の複数のメッセージの到着順序に良い非決定性を与えることも可能である。その場合でも、上記処理はＩｒ１ないしＩｒ２が複数の要素を持つことによって解決されるので、正しく動作することは明らかである。

【０２９７】また、上記例では、各プロセスないし各メソッドの処理の始めにプロセス制御部を呼び出して実現する実施例を示したが、各計算機のオペレーティングシステムが、宛先プロセス毎の処理待ちメッセージ保存部を保持し、メッセージ処理の都度、該当する分割ログ情報を参照して、該当プロセスを起動すること又は処理待ちメッセージ保存部に保存することによって実現することも可能である。

【０２９８】この場合は、図９８におけるステップＭ７で次の処理対象メッセージを送信することなく、実行待ち状態から次に起動可能なプロセスとしてスケジューリング対象に変更すれば良い。

【０２９９】本実施例によれば、ソースプログラムを修正することなく遅次化ルールであるログ情報を書き換えることにより、処理タイミングの非決定性に起因する不具合を容易に解決することができる。このことによって、処理タイミングの非決定性が内在する並行／並列／分散プログラムの開発を容易にし、生産性を向上させることができる。

【０３００】また、本実施例では、ユーザの意図する良い非決定性のみを容易に導入することが可能になるため、並行プログラムとしての柔軟性、再利用性、拡張性を維持することもできる。

【０３０１】更に、並行プログラムを遅次化して得られた全プロセスの集中ログ情報を各プロセス毎に分割し、分割されたログ情報に基づいて各プロセスを制御することによって、無害の非決定性を自然に導入し、集中ログ情報に基づく超遅次プログラム実行時と同一結果を高い処理効率で得ることができる。

【０３０２】尚、本実施例に係るコンピュータシステムの構成は、図１において、（ａ）　共有メモリ３が存在しない構成の場合、（ｂ）　Ｉ／Ｏインタフェース２がバスで密にプロセッサ１－１～１－Ｎが接続している並列計算機の場合、（ｃ）　Ｉ／Ｏインタフェースが通信路で疎に結合している分散ネットワーク計算機構成の場合、（ｄ）　単一プロセッサで構成される場合、及びそれらの組み合わせの場合、が考えられる。

【０３０３】（実施例９）本実施例で使用される新しい概念であるセクションについて説明をする。

【０３０４】本実施例におけるセクションとはスレッドの入口を１個のみ持ち、出口を複数持つプログラム断片を言う。ここでスレッドとは、コンピュータ言語の分野で一般的に用いられる用語であり、プログラムをたどりながら実行をする作用体をいう。例えばスレッドが同時に複数走った場合には、それは並行実行を意味している。

【０３０５】下記のようなプログラム断片があったとする。

```
ｃｏｄｅ１
ｃｏｄｅ２
：
ｂｒａｎｃｈ　処理１
処理２
```

ここで、簡単のためｃｏｄｅ１からｂｒａｎｃｈ命令までの区間で、区間外への分岐命令又は同期命令はないも

のとする。このプログラム断片をスレッドが処理していくとき、ｃｏｄｅ１が実行の開始点、すなわち、ｃｏｄｅ１がスレッドの入口となる。次に、処理が進みｂｒａｎｃｈ命令に到達した時、ｂｒａｎｃｈ命令までの処理内容に応じて処理１に分岐するか処理２に進むかが決まるとすれば、スレッドが処理１に進む場合と処理２に進む場合の２個の出口があることになる。従って、ｃｏｄｅ１からｂｒａｎｃｈ命令までのプログラム断片はスレッドの出口を２個持つセクションとなる。ところで、もしｃｏｄｅ１からｂｒａｎｃｈまでの間でこのプログラム断片から外部への分岐命令があった場合は、そこもこのセクションの出口の一つとなる。

【０３０６】上記の例は、１本のスレッドが複数の出口から１個の出口を選択して出ていく場合を示したが、次にプログラム断片の内部でスレッドの生成が行なわれる場合の例を示す。

```
ｃｏｄｅ１
ｃｏｄｅ２
：
ｆｏｒｋ
処理１
```

上記の例では、ｆｏｒｋ命令でスレッドの分裂、すなわちもう一つのスレッドの生成が行なわれる。この時、処理１は分裂した２本のスレッドによって並行に処理されるが、これを２個の出口から各々のスレッドが出て行き、処理１を実行するものとみなすことができる。従って、ｃｏｄｅ１からｆｏｒｋ命令の間は、この区間に外部への分岐命令がなければ、出口を２個持つセクションとなる。このように、プログラム断片の内部でスレッドの生成が行なわれる場所も一種の分岐とみなされ、複数の出口をもつセクションとなり得る。

【０３０７】図１００は、上記のようなセクションを概念的に表した図である。図１００に示すように、セクションは、スレッドの入口１０１と、セクションの本体部分１０２と、スレッドの複数の出口１０３とからなる。以後セクションを概念的に図示する時にはこのような表現を用いる。

【０３０８】次にセクションの融合と分割を説明する。図１０１はセクションの融合を表す概念図である。図１０１は、Ｓ１１とＳ１２の２個のセクションが融合してＳ１３という新たなセクションを生成する例を示している。図１０１中の矢印２０１はＳ１１の出口の一つであり、それをＳ１２の入口に接続することで２個のセクションの融合を可能としている。この例では２個のセクションの融合を示したが、一般に複数のセクションにおいて出口と入口を接続することにより、新たなセクションを生成（融合）することができる。

【０３０９】図１０２はセクションの分割を表す概念図である。図１０２は、セクションＳ２１を２個のセクションＳ２２とＳ２３に分割する例を示している。ここで、以下のプログラム断片の例を考える。

```
ｃｏｄｅ１
：
ｃｏｄｅ２
ｃｏｄｅ３
：
ｃｏｄｅ４
```

ただし、ｃｏｄｅ１～４には分岐命令を含まないものとする。これをスレッドが処理する時、ｃｏｄｅ１が入口でｃｏｄｅ４が出口となることから、このプログラム断片を出口が１個のセクションとみなすことができる。このプログラム断片をセクションＳ２１に対応させる。次に、例えば、ｃｏｄｅ２とｃｏｄｅ３の間で分割して、ｃｏｄｅ１～ｃｏｄｅ２をセクションＳ２２、ｃｏｄｅ３～ｃｏｄｅ４をセクションＳ２３に対応させれば、セクションＳ２１はセクションＳ２２とセクションＳ２３に分割されたことになる。このように１個のセクションを複数のセクションに分割することが可能である。

【０３１０】次にセクションの分類について説明する。セクションは基本セクションとグループセクションに分類される。基本セクションは、内部に分岐命令や同期命令を含まないセクションをいう。基本セクションを分割した場合には、基本セクションになる。グループセクションは基本セクション以外のセクションをいい、グループセクションは複数の基本セクションに分割することができる。

【０３１１】セクションに予測実行時間を付加する場合を説明する。ここではセクションを構成する個々の命令の実行時間（又は平均実行時間）がわかっているものとし、この時間を各命令の予測実行時間と呼ぶことにする。

【０３１２】基本セクションは途中で分岐することなく直列に実行されるので、セクションを構成する各々の命令の予測実行時間を合計することにより、そのセクションの入口から出口までの予測実行時間が計算される。基本セクションの場合は、出口が複数あってもセクションの出口は必ず命令列の最後の分岐命令又は同期命令であるため、一般に入口から各出口までの予測実行時間は等しくなる。

【０３１３】グループセクションは基本セクションに分解できるため、グループセクションの入口から各出口までに通過する基本セクションの予測実行時間を加えることにより、出口までの予測実行時間が計算される。すなわちグループセクションでは出口によって予測実行時間は変わってくる。また、グループセクション内部に処理結果に応じて回数が変わるループや無限ループ、ｊｏｉｎ等の同期命令、入出力命令を含む実行時間の予測は不可能である。その時は、グループセクションを、上記のような実行時間の予測ができないようなループ、同期命令、入出力命令を内部に含まない複数のセクションに分

割し、分割されたセクションを対象とすれば良い。

【０３１４】上記の方法によりセクションの各出口までの予測実行時間が計算された後に、その中で最大のものをセクション自身の予測実行時間とする。

【０３１５】セクションの予測実行時間は、セクションの実行の効率化のためにセクションの融合・分割をするときの指針となり、プログラム自身を最適化するときの指針としても利用できる。

【０３１６】本実施例では同期の容易さ、実行時間の予測の容易さのため、ｆｏｒｋ等のスレッドを生成する命令とｊｏｉｎ等の同期命令、入出力命令はセクションの内部には含まれずセクションの終端に位置し、実行時間の予測不可能なループは内部に持たないものとする。

【０３１７】実施例９を図面を参照して具体的に説明する。図１０３は、本発明の実施例９の概略構成を示す図である。

【０３１８】プログラム解析部４１１は、図示されない入力部から入力されたソースプログラムを解析し、基本セクションへの分割及びセクションの実行順序に関するルールの抽出をおこなう。プログラム解析部４１１で解析された情報（すなわちセクション情報）はセクション情報記憶部４１２に記憶される。

【０３１９】セクション情報編集部４１３はセクション情報記憶部４１２に記憶されたセクション情報の編集を行なう。具体的には、セクションの実行順序に関するルールの編集やセクションの融合・分割が、自動的に又はユーザによって行なわれる。コンパイル部４１４は各セクションをコンパイルしてオブジェクトコードに変換し、その結果をセクション情報記憶部４１２に登録する。

【０３２０】実行部４１５はセクション情報記憶部４１２に記憶されたセクション情報に基づき、セクションを単位として逐次又は並行にプログラムを実行していく。実行方法の逐次と並行の切替えは、逐次実行モードと並行実行モードの２種類のモード間を切替えることによって行なわれる。逐次実行モードで実行をすれば並行プログラムの再現性の問題を回避することができる。すなわち何度実行しても同じ結果が得られるため、並行プログラムについて有効なテスト・デバッグを行なうことができる。その状態で並行実行モードに切替えて実行すれば、セクション単位の同期をとりながら実行をすることにより、安全な並行実行を行なうことができる。なお、実行部４１５がインタプリタ的にのみプログラムを実行する場合は、コンパイル部４１４は省略することができる。

【０３２１】実施例９を構成する各部分について詳細に説明する。

【０３２２】まず、図１０３に示されるセクション情報記憶部４１２について、特にセクションに関する情報がどのような構造で記憶されるのかを詳細に説明する。セクション情報は、セクション自身に関する情報とセクションの実行順序に関する情報（ルール）とに２分される。まずセクション自身がどのように記憶されているかを図１０４を用いて説明する。

【０３２３】図１０４は、出口が３個のグループセクション２個と出口が２個の基本セクション１個とから構成される例を示し、全体として出口が６個のグループセクションの記憶方法を示している。セクション情報記憶部４１２は、図１０４に示すような、セクションの管理テーブル５０１、５０５、５０６、５０９を有する。これらの管理テーブルには図１０５に示すように、セクションのラベルと、セクションがコンパイルされているかを示すフラグと、コンパイルされているならそのオブジェクトを指すポインタと、セクションがプログラムコードかを示すフラグと、このセクションを構成するセクションの部品テーブルを指すか又はプログラムコードを指すポインタと、各出口を指すポインタとその出口までの予想実行時間等が記憶される。

【０３２４】セクション情報記憶部４１２は、更に、図１０４に示すセクションの部品テーブル５０２、５０３、５０４を有する。部品テーブルとはグループセクションを構成する個々の部品としてのセクションについて記憶するテーブルである。具体的には、部品テーブルは、図１０６に示すように、部品としてのセクションの管理テーブルを指すポインタと、各出口がどのセクションに接続されているかを指すポインタ等が記憶される。図１０４に示す例では部品テーブル５０２は他のグループセクションの管理テーブル５０５を指すポインタと、３個のうち２個の出口の接続先のセクションの、部品テーブル５０３、５０４を指すポインタを記憶している。ここで、例えば部品テーブルにおける出口５０８はどの部品テーブルにも接続されていない。この時、その出口を指すポインタは、管理テーブル５０１にセクションの出口として記憶される。また、管理テーブル５０６がプログラムコード５０７を指しているように、管理テーブルはプログラムコードも同様に管理している。

【０３２５】セクションの実行順序に関するルールを説明する。この場合において、スレッド１が以下のプログラム断片を実行するとする。

処理１

ｊｏｉｎ　スレッド２

処理２

上記のセクションは、スレッド１が処理１を終了させた後、スレッド２の終了を待ち、処理２を行なう、ということを意味している。図１０７は上記の例をセクションの概念図に置き換えて図示したものである。Ｓ３１は上記の処理１を構成するセクションのうち最後のセクションである。Ｓ３２は上記の処理２を構成するセクションのうち最初のセクションである。Ｓ３３はスレッド２が実行する最後のセクションである。この時、このｊｏｉ

ｎ文によって、
セクションＳ３１の実行後、スレッド２におけるセクションＳ３３の実行が終了してからセクションＳ３２の実行が開始される
というセクション実行順序のルールが抽出され、セクション情報記憶部４１２に記憶される。このようにセクションの実行順序に関するルールは、主に実行時にセクション間の同期をとるために利用される。

【０３２６】図１０８は図１０３におけるプログラム解析部４１１の詳細な構成を示す。図１０８において中間言語変換部９０１は図示されない入力部から入力されたソースプログラムを中間言語に変換する。中間言語変換部９０１で中間言語に変換するのは、セクションの抽出がしやすくなることと、各命令の実行時間が予測しやすくなるという理由による。なお、中間言語変換部９０１での変換に当たっては従来のコンパイラの技術がそのまま使用できる。セクション情報抽出部９０２は、中間言語変換部９０１によって変換された中間言語から基本セクションを切り出し、セクションの実行順序に関するルール等を抽出する。

【０３２７】プログラム解析部４１１における処理の流れを図を用いて詳細に説明する。図１０９はプログラム解析部４１１における処理の流れを示した図である。

【０３２８】入力されたソースコードは、中間言語変換部９０１において中間言語に変換される（ステップＮ１）。変換された中間言語はセクション情報抽出部９０２に引き渡される。

【０３２９】セクション情報抽出部９０２は、中間言語から基本セクションの切り出しや実行順序のルールを抽出するために初期化される（ステップＮ２）。この初期化は新しいセクションの切り出しのたびに行なわれる。この例では、基本セクションを切り出しながら同時にそのセクションの予測実行時間を計算するため、それに関する初期化も行う。

【０３３０】中間言語を１行ずつ読み込み、処理を繰り返していく。この繰り返しの中で中間言語の読み込みがすべて終了した時は（ステップＮ３）、この時点まで読み込んだ中間言語列を１個の基本セクションとしてセクション情報記憶部４１２に記憶し（ステップＮ８）、処理を終了する（ステップＮ９）。

【０３３１】中間言語の読込みが終了していないのであれば、更に中間言語を１行読込み（ステップＮ４）、読み込んだ中間言語が、例えばｊｏｉｎセクションの実行順序のルールを決める命令ならば（ステップＮ５）、それによって決定されるルールをセクション情報記憶部４１２に登録する（ステップＮ１０）。中間言語が分岐又は同期命令ならば（ステップＮ６）、その命令は基本セクションの切れ目となるので、初期化以降に読み込んだプログラム断片を１個の基本セクションとしてセクション情報記憶部４１２に記憶し（ステップＮ１１）、次のセクションの抽出を行うために、ステップＮ２に戻る。ステップＮ４において、読み込んだ中間言語の命令が上記の条件を満たさなければ、その命令の予測実行時間を切り出し中のセクションの予測実行時間に加え（ステップＮ７）、ステップＮ３に戻る。ここで中間言語の各命令の予測実行時間は、あらかじめ測定をする等してテーブル等に記憶させておけば良い。以上の手続きが終った時点で、もとのプログラムは、中間言語列として切り出された基本セクションと、セクション実行順序のルールとに変換され、セクション情報記憶部４１２に記憶されたことになる。

【０３３２】セクション情報編集部４１３の詳細な説明をする。セクション情報編集部４１３は大きく二つの動作をする。一つはセクションの融合・分割をする動作であり、もう一つの動作は、セクションの実行順序のルールを編集する動作である。

【０３３３】セクションの融合・分割の主な目的としては、セクションの大きさを揃えることによって、実行部４１５においてより効率的な実行を行なわせることにある。本実施例では、この動作にセクションの予測実行時間を利用している。セクション情報編集部４１３では、各セクションの予測実行時間がほぼ均一になるようにセクションの融合・分割をすれば良く、これは容易に自動化できる。

【０３３４】また、セクション情報編集部４１３のセクションの実行順序のルールの編集により、実行時におけるセクション間の同期や、実行順序等を自由に変更することができ、効果的なテスト・デバッグを可能とする。

【０３３５】図１１０を用いてセクション情報編集部４１３の詳細な構成を説明する。図１１０においてセクション情報変換部１１０１は、図示されない入力装置により入力されたセクションの融合・分割の指示、セクションの実行順序のルールの編集要求に従い、セクション情報記憶部４１２に記憶されたセクション情報を変更する。この動作は主にユーザによってなされるが、セクションの融合・分割に関してはセクションの予測実行時間を利用することにより容易に自動化をすることができる。セクション情報表示部１１０２はセクション情報記憶部４１２に記憶されたセクション情報を、セクション情報変更部１１０１からの指示に従い表示する。

【０３３６】図１１１は図１０３におけるコンパイル部４１４の詳細な構成を示す。図１１１において、セクション末端処理部１６０１は、セクションの末端に次に実行すべきセクションの登録とセクションの終了のためのコードを書き込む。セクションコンパイル部１６０２は、末端処理を受けたセクションをコンパイルしてオブジェクトに変換し、セクション情報記憶部４１２に登録する。

【０３３７】ここで、セクションの末端の処理の仕方を説明する。例えば、あるセクションの末端が以下のよう

なコードであったとする。
処理１
ｂｒａｎｃｈ　ｌａｂｅｌ
処理２
ｌａｂｅｌ：
処理３
上記のコードは、処理１の結果によって、ｌａｂｅｌにジャンプして処理３を実行するか、そのまま処理２を実行する、ということを表したコードである。ここで、処理２の先頭に位置するセクションをセクション２、処理３の先頭に位置するセクションをセクション３、としたとき、上記コードは以下のように処理される。
処理１
ｂｒａｎｃｈ　ｌａｂｅｌ
ｓｅｃｔｉｏｎ－ｅｎｄ　セクション２
ｌａｂｅｌ：
図１１２は図１０３における実行部４１５の詳細な構成図を示す。図１１２において、実行セクション選択部１２０１は、セクション情報及び実行セクション候補記憶部１２０３に記憶される情報に基づき、次に実行すべきセクションを１個又は複数選択する。実行セクション選択部１２０１は、図示されない入力部から切替えることが可能な２つのモード（逐次実行モードと並行実行モード）を有する。逐次実行モードではセクションの選択は１個である。並行実行モードではセクションの選択は一般に複数となる。セクション実行部１２０２は、実行セクション選択部１２０１によって選択されたセクションを実行する。特に、選択されたセクションが複数の場合は並行に実行する。実行セクション候補記憶部１２０３は、セクション実行部１２０２で実行した結果から決められる次の実行の候補となるセクションを記憶する。
【０３３８】図１１３は、実行部４１５における処理の流れを示す図である。プログラムは、セクション情報記憶部４１２に記憶されるセクションの情報及び実行順序のルールに従って、セクション単位に実行される。
【０３３９】全体の初期化が行なわれる（ステップＰ１）。この時、実行セクション候補記憶部１２０３には最初に実行すべきセクションが記憶される。
【０３４０】次にセクション毎に実行を行なうループに入る。このループの最初では、次に実行するセクションの選択が、モードによって異なる方法で行なわれる（ステップＰ２）。逐次実行モードでは、実行セクション候補記憶部１２０３に記憶される次に実行されるセクションの候補の中から実行順序のルールに従うものを１個選択する（ステップＰ３）。一般に候補となるセクションの中には、ルールに従うものが複数ある。これから１個を選択する方法としては、例えば記憶された時間が最も前のものを選択しても良いし、又は、固定された乱数列に従って選択しても良い。ただし、ここでの選択方法としては再現性が保証される方法を用いる必要がある。プログラムの実行のたびに選択される順序が変化しない（すなわち再現性が保証されている）ことは本発明の重要な動機の一つだからである。モードが並行実行モードでは、実行候補の中からルールに従うセクションをすべて選択する（ステップＰ４）。選択されたセクションは一般に複数となる。
【０３４１】ステップＰ３又はステップＰ４において、選択に失敗した場合は（ステップＰ５）、プログラムが終了したか、実行順序を決めるルールの異常により失敗した場合が考えられる。そこで、候補のリストを確かめ（ステップＰ６）、候補が残っていないならば正常終了（ステップＰ７）、残っているならば異常終了（ステップＰ８）とする。異常終了の場合には、例えばデッドロック状態が考えられ、この旨をユーザに提示することによりテスト・デバッグの指針とすることができる。ステップＰ３及びステップＰ４において、選択に成功した時は、選択されたセクションの実行をする（ステップＰ９）。
【０３４２】複数のセクションが選択された時には、それらのセクションは並行に実行される。この後セクション実行部４１５は、並行に実行されたセクションすべての実行の終了を待ってから、次のステップに進む。
【０３４３】なお、本実施例で使用しているセクションの内部には、無限ループ等の実行時間を予測不可能にする因子は含まれていないため、セクションの実行終了を永久に待ち続けるということはない。
【０３４４】実行が終了すると、まず終了したセクションが実行セクション候補記憶部１２０３から消去され、次に実行結果に応じて次の実行候補となるセクションが記憶される（ステップＰ１０）。「実行結果に応じて」ということは例えばセクションの最後がｉｆ文のときは、計算結果によって次に分岐する分岐先、すなわち次に実行するべきセクションが変わる、ということを意味する。また、セクションがｅｎｄ文で終了するときは、それを実行していたスレッドも終了するので、このスレッドが次に実行すべきセクションは無いことになる。この時は、実行セクション候補記憶部１２０３には何も記憶されない。
【０３４５】実行セクション候補記憶部１２０３での記憶内容の詳細な変化を説明する。図１１４は、プログラムの一例をセクションの概念図を使って表した図である。１本のスレッドがセクション１を実行し、その中のｆｏｒｋ文により２本のスレッドに分裂し、それぞれのスレッドがセクション２とセクション４を実行し、ｊｏｉｎ命令により同期をとって１本のスレッドに戻り、最後にセクション３を実行して終了する例を示す。ここでｊｏｉｎ文により同期をとるために、セクション３はセクション４の後に実行されるというセクションの実行順序に関するルールが、事前のプログラム解析により抽出されているとする。

【0346】図115は上記の例を、逐次実行モードで実行した時の実行セクション候補記憶部1203の変化を示す図である。

【0347】初期状態1401では最初に実行されるセクション1が記憶されている。そこで、まずセクション1が実行される。セクション1が実行されるとfork文によりスレッドが2本に分裂することから、次の実行候補としてセクション2とセクション4の2セクションが記憶される（状態1402）。ここで並行実行モードであればセクション2とセクション4が同時に選択され実行されるが、ここでは逐次実行モードで実行しているため、どちらか1個が選択されなければならない。そこでセクション2を選択し、実行したとする。セクション2が終了すると、次の候補としてのセクション3が記憶される（状態1403）。

【0348】次に、候補を見る限りではセクション3とセクション4のどちらを選択しても良さそうだが、先に示した実行順序のルール「セクション3はセクション4の後に実行される」により、セクション4しか選択することができないので、セクション4が実行されることになる。セクション4の実行が終了すると、それを実行したスレッドが終了するため候補記憶部1203ではセクション4に続くセクションが無くなっている（状態1404）。そして、セクション3を実行し全処理を終了したことになる（状態1405）。

【0349】本発明は、上記実施例に限定されない。

【0350】上記実施例は、それぞれ独立の実施例として記載したが、それぞれ適宜組み合わせることにより、より効果的な並行プログラムを作成することが可能になる。特に、実施例1～実施例8において作成された並行プログラムを実施例9の並行プログラム実行装置で実行させてその妥当性を検証することも可能であるし、実施例9において、ソースプログラムである並行プログラムを並行プログラム実行装置で実行させて、実施例1～実施例8に示すように、実行情報（テスト・デバッグ情報を含む）に基づいて、逐次プログラムを並行化する、或いは、ソースプログラムをバグのない並行プログラムに変換しても良い。

【0351】本実施例によれば、実行ログを意味を変えない範囲でユーザに理解し易い形に並べ替え、ユーザに提示することができる。並べ替えても意味が変わらないことが保証されているので、オリジナルの実行ログにバグがある場合は、並べ替え後の実行ログにおいてもバグが存在する。この性質により、テスト・デバッグの効率が向上するという効果が得られる。

【0352】（実施例10）本実施例では、図2の逐次化装置12の具体的な実現法について述べる。図117は逐次化装置12の実施例を示すブロック図である。この逐次化装置12は、テスト実行装置121、第1実行ログファイル記憶部122、解析装置123、先行関係情報ファイル124、並べ替え装置125及び第2実行ログファイル記憶部126からなる。

【0353】図118は図117の逐次化装置12を用いた場合の並行プログラム作成方法の概略手順を示すフローチャートである。

【0354】（1）　ステップA1：モデル化

対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、各プロセス構造を決定する。更に、該プロセス内を並行プログラム等を用いたプログラミングにより、ソースプログラムCPを記述する。このソースプログラムには、バグが潜在的に存在する可能性がある。図119は、こうして記述されたソースプログラムCPの具体例であり、2つのプロセスP1とP2をプログラミングしている。

【0355】（2）　ステップQ1：実行

テスト実行装置121により、ソースプログラムCPを実行し、実行ログを第1実行ログファイル記憶部122に格納するとともに図2の出力装置5に表示する。この例の場合、図119のソースプログラムCPに対する可能な実行ログは数多く存在するが、ここでは、図120のような実行ログが生成されたとする。しかし、この実行ログはプロセスP1とプロセスP2が複雑に絡み合っており、実行過程が把握しにくいものとなっている。

【0356】（3）　ステップQ2：バグの判定

テストの結果、バグがなければ終了する。テストの結果バグがあれば、その実行ログ3を保存しステップQ3に進む。この例の場合、図120の実行ログが示す実行後のMの値は「M＝－1」であり、バグであると判定したとする（期待した値は「M＝0」）。

【0357】（4）　ステップQ3：解析

解析装置123により、第1CPファイル記憶部11からのソースプログラムCP及び第1実行ログファイル記憶部122に格納された実行ログからプロセス間の先行関係情報を抽出し、先行関係情報ファイル124に格納する。図121に、図119の実行ログにおける2つのプロセスP1とP2の各命令の先行関係を示す。

【0358】（5）　ステップQ4：実行ログの並べ替え

第1実行ログファイル記憶部122と先行関係情報ファイル124に格納された実行ログ及び先行関係情報から、並べ替え装置125によりユーザが理解しやすい順序に並べ替えた実行ログを生成し、第2実行ログファイル記憶部126に格納する。この例の場合、図121の先行関係を満たす範囲で、ユーザが理解しやすい順序に並べ替えた実行ログを生成する。具体的な並べ替え規則の一例としては、（a）プロセスに優先順位を導入し、その優先順位を用いて実行ログを並べ替える、（b）待ち状態が解除されたプロセスを優先して実行ログを並べ替える

（c）ユーザが指定する、などがある。図122に、プ

ロセスに優先順位を導入して並べ替えた後の実行ログを示す。

【0359】（6）　ステップQ5：実行ログの表示
第2実行ログファイル記憶部126に格納された並べ替え後の実行ログを逐次プログラムとして出力し、かつ出力装置5により表示してユーザに提示する。

【0360】この例では、図122に示す実行ログが出力装置5で図122に示すように表示される。この実行ログはプロセスP1とプロセスP2の命令が先行関係を保つ範囲でまとまっており、ユーザにとって実行過程が把握し易いものとなっている。

【0361】（7）　ステップQ6：並行プログラムのデバッグ／修正
ユーザは出力装置5で表示された実行ログを見て実行過程を把握又は実行を再現し、バグの原因を発見すればプログラムを修正する。プログラムを修正したらステップQ1に戻る。この例の場合、図122の実行ログの表示から、ユーザは、バグの原因は「P2：read（M、Y);」と「P1：write(M、XX);」の順序が間違っていることであると認識し、P2のｓｅｎｄ命令の場所を修正することができる。修正されたプログラムは図123のようになる。この修正後のプログラムを実行すると、実行後のMの値は「M＝0」となり、バグが削除できたことがわかる。

【0362】（実施例11）本実施例では、テスト実行は部分超逐次プログラムに対して行うが、ソースコードの修正はオリジナルの並行プログラムに対して行う。また、良い非決定性に関する情報の導入は、逐次化条件に対して行い、良い非決定性の導入毎にオリジナルの並行プログラムの逐次化（部分逐次化）を行う。

【0363】図124は、本実施例に係る並行プログラム作成支援装置の概略構成を示す図である。同図において、第1CPファイル記憶部11に格納されたソースプログラムCPは、ユーザによる入力装置4からの指示により引き出され、逐次化装置12に入力される。逐次化装置12に入力されたソースプログラムCPは、逐次化ルール記憶部13に格納された逐次化ルールに従って超逐次プログラムHSP又は部分超逐次プログラムPHSPに変換され、HSP／PHSPファイル記憶部141に記録される。

【0364】このプログラムHSP／PHSPに対して、テスト実行装置15でテストが行われる。ここで、バグがあればデバッグ装置16を用いて、第1CPファイル記憶部11に格納されているソースプログラムCPを修正する。修正されたソースプログラムCPは再度逐次化され、超逐次プログラム（HSP）又は部分超逐次プログラム（PHSP）が生成されてHSP／PHSPファイル記憶部141に記録される。

【0365】この一連のサイクルを繰り返し、テスト・デバッグが完了したならば、非決定性導入装置17で、逐次化ルール記憶部13に格納されている逐次化ルールの制約を緩めて、部分超逐次プログラムPHSPを生成する。そして、この部分超逐次プログラムPHSPに対して、同様のテスト・デバッグを繰り返しながら、非決定性導入装置17で、徐々に逐次化ルールの制約を弱めていく。このようにして良い非決定性導入が完了したならば、最終的に並行化装置18で並行プログラムが生成され、第2CPファイル記憶部19に格納される。

【0366】（実施例12）本実施例では、テスト超逐次プログラムに対して行うが、バグのあった場合はソースコードたる並行プログラムを修正せずに、逐次化ルールを修正することでテスト・デバッグを行う。その他の部分に関しては、基本的にこれまでの実施例と同じである。

【0367】本実施例における並行プログラムの作成は、以下のような手順によって実現される。図125は、本実施例に係る並行プログラム作成方法の概略手順を示すフローチャートである。

【0368】（1）　ステップA1：モデル化
対象の並行システムに対し、並行性を用いた自然なモデル化を行う。また、各プロセス構造を決定する。更に、該プロセス内を並行プログラム等を用いたプログラミングにより、並行構造を有する並行プログラムをソースプログラムとして記述する。このソースプログラムには、バグが潜在的に存在する可能性がある。

（2）　ステップA2：逐次化
ステップA1で得られた並行プログラムを所定の逐次化ルールに基づいて超逐次プログラムに変換する。

【0369】（3）　ステップA3：テスト・デバッグ
ステップA2で変換された超逐次プログラムを実行してテストを行い、機能面のバグがある場合は、超逐次プログラムを修正してテスト・デバッグを行う。

【0370】（4）　ステップC2：バグ判定
ステップA3のテストの結果、タイミング面のバグがある場合は、ステップJ1に移行する。いずれのバグもない場合は、その超逐次プログラムを保存しステップA4に移行する。

【0371】（5）　ステップJ1：逐次化ルールの修正
ステップC2でタイミング面のバグがあると判定された場合は、逐次化ルールを修正し、ステップA2に戻る。

【0372】（6）　ステップA4：良い非決定性の導入
良い非決定性の追加が必要な場合は、ステップJ1により逐次化ルールを修正して新たな逐次化ルールを設定し、ステップA2に戻る。良い非決定性の追加が必要でない場合は、ステップA7に移行する。

【0373】（7）　ステップA7：並行化コンパイル
保存された超逐次プログラムの集合から、無害な非決定性部分を抽出し、それを並行プログラムとして生成す

る。

【０３７４】図１２６は、本実施例に係る並行プログラム作成支援装置の概略構成を示す図である。同図において、第１ＣＰファイル記憶部１１に格納されているソースプログラムＣＰは、逐次化装置１２において逐次化ルール記憶部１３に格納された逐次化ルールに従って、メタレベルにおいてデフォルト逐次性が導入されることにより、超逐次プログラム（ＨＳＰ）に変換され、ＨＳＰファイル記憶部１４に記録される。ユーザは、この超逐次プログラムＨＳＰを出力装置５によって見ることができる。この超逐次プログラムＨＳＰは、ユーザによる入力装置４からの指示によってテスト実行装置１５に入力され、テスト実行が行われる。

【０３７５】テスト実行装置１５は、テスト実行の結果（実行ログ）を出力装置５に提示する。ユーザは、このテスト実行の結果に基づいて、逐次化ルール修正装置２６を用いて逐次化ルール記憶部１３に格納されている逐次化ルールを修正し、超逐次プログラムＨＳＰに対し所定のテスト・デバッグを行うことができる。

【０３７６】ユーザは、このように逐次化ルールの修正により超逐次プログラムＨＳＰに対して所定のテスト・デバッグを行った後、入力装置４から再度テスト実行の指示を与える。これにより、テスト・デバッグの行われた超逐次プログラムＨＳＰは、テスト実行装置１５に入力され、再度テスト実行が行われる。このテスト・デバッグは、超逐次プログラムＨＳＰが正常に動作することを確認するまで繰り返し行われる。

【０３７７】そして、本実施例ではテスト・デバッグにより正常に動作することを確認した時点で、逐次化ルール修正装置２６により逐次化ルール記憶部１３に格納されている逐次化ルールが修正され、この修正の後の逐次化ルールが逐次化ルール記憶部１３に新たに記録される。この逐次化ルールの修正によって、超逐次プログラムに対して良い非決定性に関する情報が部分的に導入される。

【０３７８】その他本発明の要旨を変更しない範囲で種々変形して実施できるのは勿論である。

【０３７９】

【発明の効果】本発明によれば次のような効果が得られる。

【０３８０】本発明では並行プログラムを一旦逐次化し、逐次化したプログラムに対してテスト・デバッグを行うことにより、従来の並行プログラムのプログラミングより遥かに容易な逐次プログラミングと同じレベルの困難さで並行プログラムのテスト・デバッグが可能となる。

【０３８１】上記のグラフ情報によって並行化及び逐次化の情報をユーザに対して同時に提示することにより、ユーザは第１並行プログラムの並行構造を考慮しつつ、良い非決定性部分の指定をすることができるようになる。また、並行プログラム記述レベルにおける良い非決定性部分の指定・解除ではなく、グラフ情報に対して良い非決定性部分を指定・解除を行うことでができるため、高度な並行プログラミング技術を必要とすることなく、容易に並行プログラムの開発をすることができるようになる。

【０３８２】第１並行プログラムを逐次プログラムに変換する際、第１並行プログラム及びその実行ログを解析し、この解析結果に基づいて実行ログを並べ替えるようにすれば、並べ替え後の実行ログを表示してユーザに提示することによって、並行プログラムの実行過程の理解が容易となり、テスト・デバッグの効率が向上する。逐次プログラムに並行性に関する情報を導入する際、逐次プログラムのプロセスの流れを制約と遷移条件からなるフィールドに変換し、更にそのフィールドデータを表示することによって、フィールドを対話的・視覚的に編集することで並行性に関する情報を効果的に導入し、バグのない並行プログラムが効率的に作成される。

【０３８３】第１並行プログラムから逐次化された逐次プログラムに対して並行化の候補となるプロセス群を指定し、このプロセス群の実行順序を入れ換えて逐次プログラムを複数の並行模擬プログラムに変換した後、これら複数の並行模擬プログラムを部分的に逐次構造を有する１つの部分逐次プログラムに変換し、この部分逐次プログラムを並行化して第２並行プログラムに変換することにより、部分逐次プログラム上で並行プログラムの動作を十分に確認することができる。また、部分逐次プログラムに対して並行化の候補となるプロセス群を指定することで、逐次構造プログラムを段階的に並行プログラムへ変換することができる。更に、並行模擬動作系列に基づく逐次実行で正しく動作することが確認された非決定性のみを許容する並行性に関する情報を導入することで、正しく動作する並行プログラムを得ることができる。これらによって、並行プログラムのテスト・デバッグが容易となる。

【０３８４】プロセス群がメッセージ情報を交換しながら並行して動作する実行環境に用いられる並行プログラムの作成を支援する並行プログラム作成支援装置において、第１並行プログラムのプロセス群の実行履歴であるログ情報を逐次化ルールとして取得して記憶し、このログ情報を修正可能とするとともに、記憶されているログ情報に基づいてこれに基づいてプロセス群を逐次的に起動制御し、記憶されたログ情報を並行化して第２並行プログラムに変換することにより、ソースプログラムとしての並行プログラムを修正することなく、ログ情報の修正で処理タイミングの非決定性に起因する不具合を解決できる。これにより、処理タイミングの非決定性が内在する並行／並列／分散プログラムの開発が容易となる。また、ユーザの意図する良い非決定性のみを容易に導入することができるため、並行プログラムとしての柔軟

性、再利用性及び拡張性を維持することもできる。

【０３８５】プロセス群がメッセージ情報を交換しながら並行して動作できる実行環境でプロセス群が実行順序規定情報に従って動作するシステムにおいて、実行順序規定情報を分割し、つまり並行プログラムを逐次化して得られた全プロセスの集中ログ情報を各プロセス毎に分割し、この分割された実行順序情報に基づいてプロセス群を起動制御することにより、無害の非決定性を自然に導入し、集中ログ情報に基づく逐次プログラムの実行時と同一結果を高い処理効率で得ることが可能となる。

【０３８６】第１並行プログラムをテスト実行してそのテスト実行の結果、その１つであるバグのない実行ログを蓄積し、この蓄積されたバグのない実行ログのみを並行化して第２並行プログラムに変換することにより、テストで通過したタイミングだけを許容するようにプログラムが動くようになるため、テストしなかったことで残存したバグに陥ることを回避でき、信頼性が向上する。

【０３８７】本発明の並行プログラム実行装置によれば並行プログラムの逐次又は部分並行実行を行なうことが可能となるので、従来の逐次プログラムと同様に再現性を保証することができ、効果的かつ安定したテスト・デバッグを行なうことができる。また、逐次実行から部分並行実行に容易に切替えることができるため、逐次化してテスト・デバッグをおこなったプログラムを非決定性に影響されない安全な並行実行を可能とする。他に、セクションの実行順序に関するルールを編集して実行時のセクション間の同期を制御することにより、効果的なテスト・デバッグを行なえるとともにプログラムの効率化の指針とすることもできる。

【図面の簡単な説明】

【図１】　本発明に係る並行プログラム作成支援装置を実現するためのコンピュータシステムの構成図。

【図２】　実施例１に係る並行プログラム作成支援装置の概略構成を示すブロック図。

【図３】　実施例１に係る並行プログラム作成方法の概略手順を示すフローチャート。

【図４】　実施例１の動作説明のための並行プログラムの一例と逐次化ルールを示す図。

【図５】　実施例１の動作説明のためのメタレベルのデフォルト逐次性が導入された超逐次プログラムの概念図。

【図６】　実施例１の動作説明のためのデバッグ画面の一例を示す図。

【図７】　実施例１の動作説明のための非決定性に関する情報の導入例を示す図。

【図８】　実施例２に係る並行プログラム作成支援装置の概略構成を示すブロック図。

【図９】　実施例２に係る並行プログラム作成方法の概略手順を示すフローチャート。

【図１０】　セクションの設定方法の一例を示す図。

【図１１】　逐次化方式の一例を示す図。

【図１２】　逐次化情報の書換ルール例を示す図。

【図１３】　並行プログラムの例を示す図。

【図１４】　並行プログラムの記載例を示す図。

【図１５】　それぞれ、超逐次プログラムのセクション情報、プログラム構造情報、超逐次化情報を示す図。

【図１６】　良い並行性の導入を説明するための図。

【図１７】　自動並行化を行った逐次プログラミングから生成された並行プログラムのソースコードを示す図。

【図１８】　超逐次プログラムのプログラム構造情報を示す図。

【図１９】　超逐次プログラムの逐次化情報を示す図。

【図２０】　自動並行化された超逐次プログラムの処理の流れを示す図。

【図２１】　実施例３に係る並行プログラム作成支援装置の概略構成を示すブロック図。

【図２２】　実施例３に係る並行プログラム作成方法の概略手順を示すフローチャート。

【図２３】　実施例３の動作説明のためのソースプログラムである並行プログラムの一例を示す図。

【図２４】　実施例３における並行プログラム合成例を示す図。

【図２５】　実施例４に係る並行プログラム作成法の概略手順を示すフローチャート。

【図２６】　実施例４で用いるプロセステーブルを示す図。

【図２７】　実施例４の動作説明のための並行プログラムの一例を示す図。

【図２８】　図２７における並行プログラムの並行構造を示す概念図。

【図２９】　実施例４で用いる並行プログラムの構造解析の結果作成されるプロセステーブルを示す図。

【図３０】　実施例４で用いる超逐次グラフの一例を示す図。

【図３１】　実施例４における超逐次グラフ生成装置の動作処理を説明するための図。

【図３２】　実施例４における並行化部分の指定方法を説明するための図。

【図３３】　実施例４における並行化部分指定後の超逐次グラフの一例を示す図。

【図３４】　グループ化されたノードを持つ超逐次グラフの一例を示す図。

【図３５】　優先順位変更後の超逐次グラフの一例を示す図。

【図３６】　３プロセス間の良い非決定性部分の導入を説明するための図。

【図３７】　良い非決定性部分の導入後の超逐次グラフを示す図。

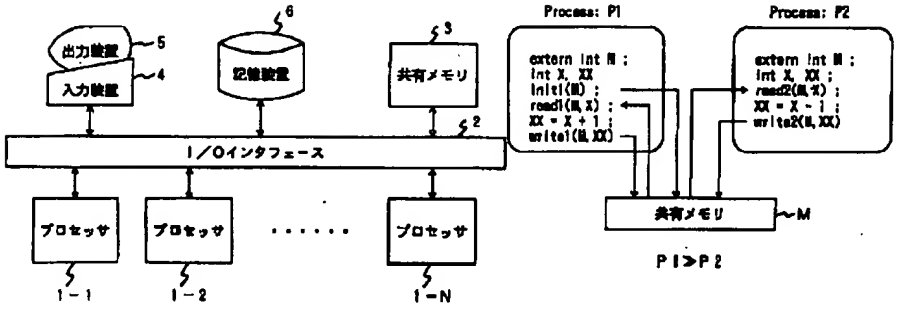
【図３８】　実施例５に係る並行プログラム作成支援装置の概略構成を示すブロック図。

【図３９】　実施例５の動作説明のためのソースプログラムである並行プログラムを示す図。
【図４０】　図３９の並行プログラムで実行される処理の流れを示す図。
【図４１】　実施例５における遅次化プロセスファイルに格納されている遅次化プロセスの一例を示す図。
【図４２】　図４１の遅次化プロセスで実行される処理の流れを示す図。
【図４３】　実施例５におけるフィールドデータ生成部で実行される処理の流れを示すフローチャート。
【図４４】　図４３の処理により生成されるエリアのデータ構造を示す図。
【図４５】　実施例５におけるフィールドデータ生成部で生成されるフィールドデータの一例を示す図。
【図４６】　実施例５におけるフィールドチューニング部で実行される処理の流れを示すフローチャート。
【図４７】　図４６の一部（ステップＥ９）を詳細に示す図。
【図４８】　実施例５における制約変更操作によって起こるフィールドの変化を示す図。
【図４９】　図４６の他の一部（ステップＥ１１）を詳細に示す図。
【図５０】　実施例５における制約変更操作によって起こるフィールドの変化を示す図。
【図５１】　図４６の更に他の一部（ステップＥ１３）を詳細に示す図。
【図５２】　実施例５における制約変更操作によって起こるフィールドの変化を示す図。
【図５３】　フィールドに矛盾を発生させる自明な制約を検出する処理を示す図。
【図５４】　自明制約を含むフィールド例を示す図。
【図５５】　自明制約の検出例を示す図。
【図５６】　フィールドエディタの表示画面の例を示す図。
【図５７】　フィールドエディタの表示画面の他の例を示す図。
【図５８】　フィールドエディタの表示画面の更に他の例を示す図。
【図５９】　実施例５におけるフィールドデータのチューニング過程の様子示す図。
【図６０】　実施例５におけるフィールドデータのチューニング過程の様子示す図。
【図６１】　実施例５におけるフィールドデータのチューニング過程の様子示す図。
【図６２】　実施例５におけるフィールドデータのチューニング過程の様子示す図。
【図６３】　実施例５におけるフィールドデータのチューニング過程の様子示す図。
【図６４】　実施例５におけるフィールドデータのチューニング過程の様子示す図。
【図６５】　実施例５におけるフィールドデータのチューニング過程の様子示す図。
【図６６】　実施例５におけるフィールドデータのチューニング過程の様子示す図。
【図６７】　実施例５におけるフィールド変換部で実行される処理を示すフローチャート。
【図６８】　実施例５における修正後の並行プログラムの一例を示す図。
【図６９】　図６８の並行プログラムで実行されるプロセスの流れをイメージした図。
【図７０】　実施例６に係る並行プログラム作成支援装置の概略構成を示す図。
【図７１】　実施例７に係る並行プログラム作成方法の概略手順を示すフローチャート。
【図７２】　実施例７の動作説明のためのユーザが想定した並行プログラムモデルの一例を示す図。
【図７３】　図７２の並行プログラムから得られる超遅次プログラムの実行系列をモデル化した例を示す図。
【図７４】　実施例７における超遅次プログラムから部分並行プログラムへの変換手順を示すフローチャート。
【図７５】　実施例７における超遅次プログラムから部分並行プログラムへの変換例を示す図。
【図７６】　実施例７における超遅次プログラムから部分並行プログラムへの変換例を示す図。
【図７７】　実施例７における超遅次プログラムから部分並行プログラムへの変換例を示す図。
【図７８】　実施例７における超遅次プログラムから部分並行プログラムへの変換例を示す図。
【図７９】　実施例８に係るプロセス間の情報交換の一例を示す図。
【図８０】　実施例８に係る並行プログラム作成支援装置の概略構成を示すブロック図。
【図８１】　実施例８に係る並行プログラム作成方法の概略手順を示すフローチャート。
【図８２】　実施例８におけるメッセージの実行履歴であるログ情報の一例を示す図。
【図８３】　実施例８に係るログ情報に基づく並行プログラムデバッグ装置の概略構成を示すブロック図。
【図８４】　実施例８におけるメッセージフォーマットの一例を示す図。
【図８５】　実施例８におけるプロセス間情報交換を示す図。
【図８６】　実施例８におけるログ情報の一例を示す図。
【図８７】　実施例８におけるプロセス群制御部の処理手順を示すフローチャート。
【図８８】　実施例８におけるログ情報を修正するために入れ替え対象を指示している一例を示す図。
【図８９】　実施例８における修正後のログ情報の一例を示す図。

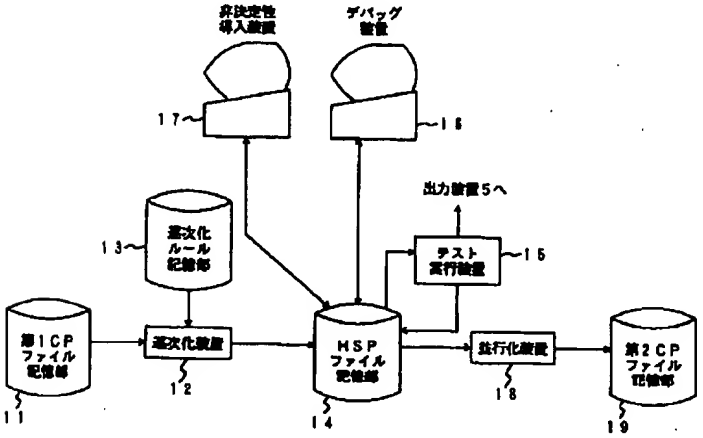
【図９０】　実施例８におけるログ情報に良い非決定性を導入するためにその対象箇所を指定している一例を示す図。
【図９１】　実施例８における良い非決定性導入後のログ情報の一例を示す図。
【図９２】　実施例８におけるログ情報に基づく並行プログラム実行システムの構成を示す図。
【図９３】　実施例８におけるタスク間情報交換を示す図。
【図９４】　実施例８における良い非決定性を導入した場合のログ情報の一例を示す図。
【図９５】　実施例８におけるタスク制御部の処理手順を示すフローチャート。
【図９６】　実施例８におけるタスク制御部を起動するための一手段であるプログラム例を示す図。
【図９７】　実施例８におけるメッセージフォーマットを示す図。
【図９８】　実施例８におけるタスク制御部の処理手順を示すフローチャート。
【図９９】　実施例８における各タスクの構成例を示す図。
【図１００】　セクションの概念を示す図。
【図１０１】　セクションの融合を示す概念図。
【図１０２】　セクションの分割を示す概念図。
【図１０３】　本発明の実施例９の概略構成を示す図。
【図１０４】　セクションを記憶するためのデータ構造を示す図。
【図１０５】　セクション管理テーブルの構造を示す図。
【図１０６】　部品テーブルの構造を示す図。
【図１０７】　セクションの実行順序のルールを説明するための図。
【図１０８】　プログラム解析部の詳細構成を示す図。
【図１０９】　プログラム解析の流れを示すフローチャート。
【図１１０】　セクション情報編集部の詳細構成を示す図。
【図１１１】　コンパイル部の詳細構成を示す図。
【図１１２】　実行部の詳細構成を示す図。
【図１１３】　実行部の処理の流れを示すフローチャート。
【図１１４】　実行セクション候補記憶部の変化を説明するための図。
【図１１５】　実行セクション候補記憶部の変化を示す図。
【図１１６】　従来技術の問題点を説明するための図。
【図１１７】　実施例１０に係る逐次化装置の構成を示すブロック図。
【図１１８】　実施例１０に係る並行プログラム作成方法の概略手順を示すフローチャート。
【図１１９】　実施例１０の動作説明のためのソースプログラムである並行プログラムの一例を示す図。
【図１２０】　実施例１０の動作説明のための図１１９の並行プログラムの実行ログを示す図。
【図１２１】　実施例１０の動作説明のための並行プログラムの先行関係を示す図。
【図１２２】　実施例１０の動作説明のための並行プログラムの実行ログの並べ替えた後の状態を示す図。
【図１２３】　実施例１０の動作説明のための修正後の並行プログラムの一例を示す図。
【図１２４】　実施例１１に係る並行プログラム作成支援装置の概略構成を示す図。
【図１２５】　実施例１２に係る並行プログラム作成方法の概略手順を示すフローチャート。
【図１２６】　実施例１２に係る並行プログラム作成支援装置の概略構成を示すブロック図。
【符号の説明】
１～１～１－Ｎ…プロセッサ　２…Ｉ／Ｏインタフェース
３…共有メモリ　４…入力装置
５…出力装置　６…記憶装置
１１…第１ＣＰファイル記憶部　１２…逐次化装置
１３…逐次化ルール記憶部　１４…ＨＳＰファイル記憶部
１５…テスト実行装置　１６…デバッグ装置
１７…非決定性導入装置　１８…並行化装置
１９…第２ＣＰファイル記憶部
２０…解析情報記憶部　２１…並行化ルール記憶部
２２…実行装置　２３…分割基準指定装置
２５…変更装置　２６…逐次化ルール修正装置
３１…逐次化情報記憶部　３２…逐次化情報読込部
３３…並行構造解析部　３４…逐次構造解析部
３５…画像データ生成部
５１…逐次化プロセスリスト記憶部
５２…フィールドデータ記憶部　５３…フィールドデータ生成部
５４…フィールドチューニング部
５５…フィールドエディタ
２０１…ログ情報記憶部　２０２…ログ情報取得部
２０３…プロセス群制御部　２０４…メッセージ保存部
２０５…ログ情報修正部
３０１…集中ログ情報記憶部　３０２…ログ情報分割部
３０３…分割基準指定部　３０４…分割ログ情報

記憶部
３０５…プロセス制御部　３０６…処理待ちメッセージ保存部
４０１…テスト実行装置　４０２…テストケース記憶部
４０３…実行ログ記憶部　４０４…実行ログデータベース
４１１…プログラム解析部　４１２…セクション情報記憶部
４１３…セクション情報編集部　４１４…コンパイル部
４１５…実行部

【図１】

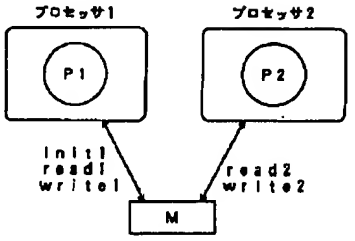


【図２】

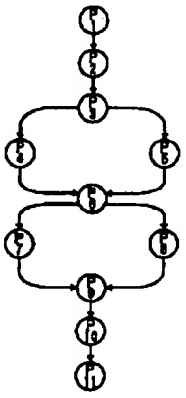


【図４】

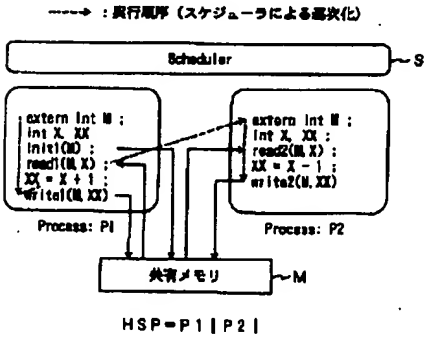
【図１３】



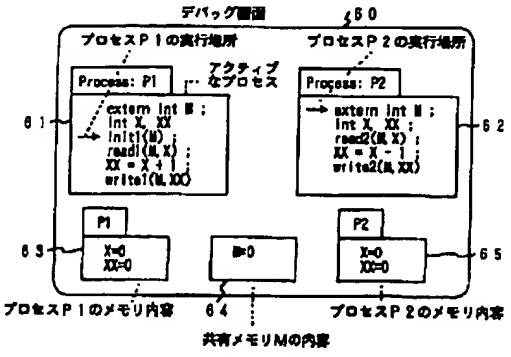
【図４０】



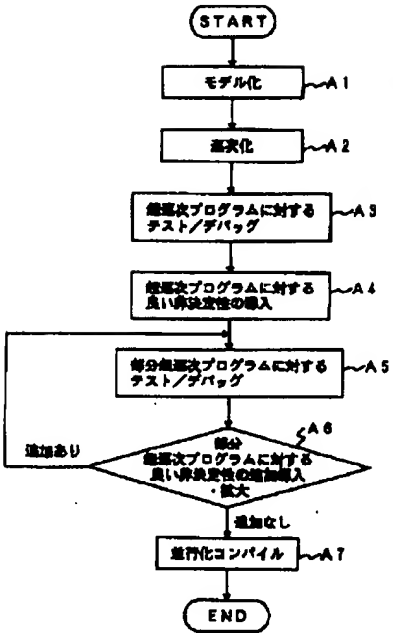
【図５】



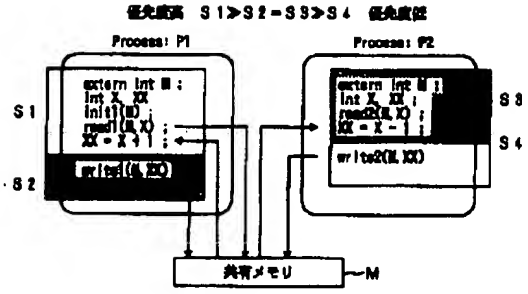
【図６】



【図３】



【図７】

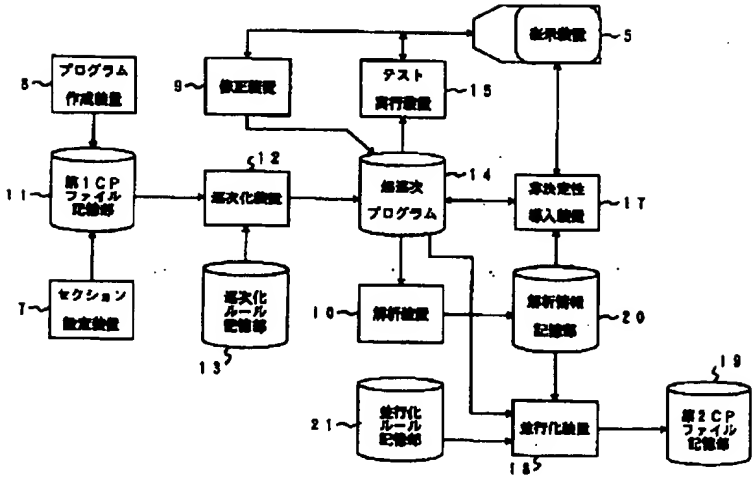


【図１４】

```
P1:
begin
init1;      /*メモリMを初期化する*/
read1;      /*メモリMから読み込む*/
write1;     /*メモリMに書き込む  */
end

P2:
begin
read2;      /*メモリMから読み込む*/
write2;     /*メモリMに書き込む  */
end
```

【図８】

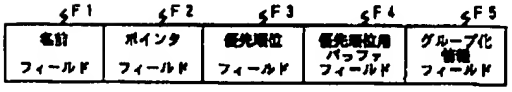


【図１７】

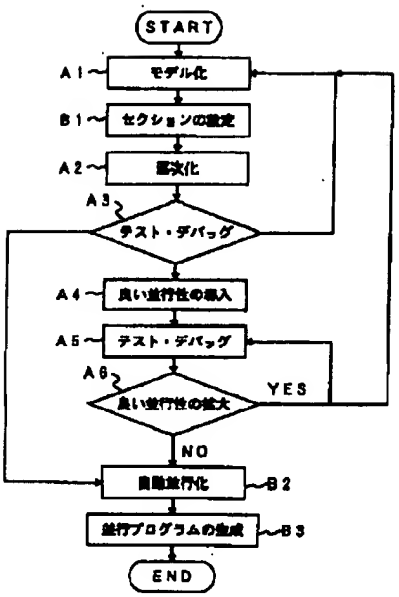
```
P1:
begin
init1;      /*メモリMを初期化する*/
send(1);    /*セマフォ  */
read1;      /*メモリMから読み込む*/
write1;     /*メモリMに書き込む  */
send(2);    /*セマフォ  */
end

P2:
begin
wait(1);    /*セマフォ  */
read2;      /*メモリMから読み込む*/
wait(2);    /*セマフォ  */
write2;     /*メモリMに書き込む  */
end
```

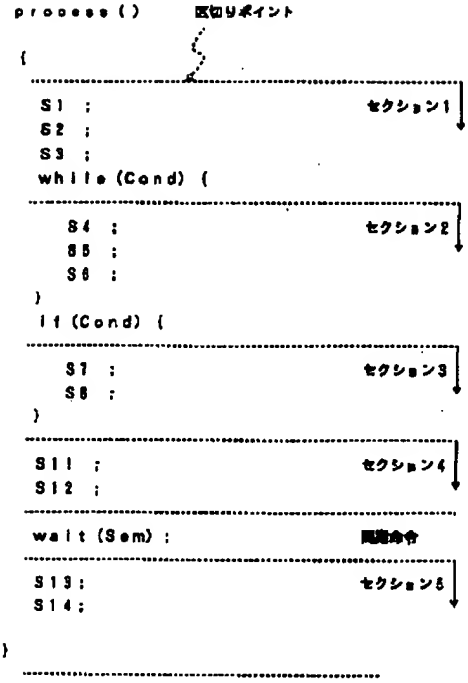
【図２６】



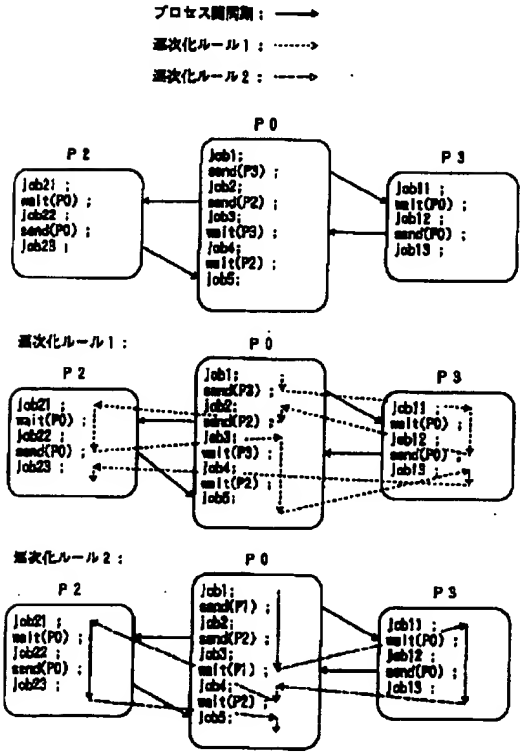
【図9】



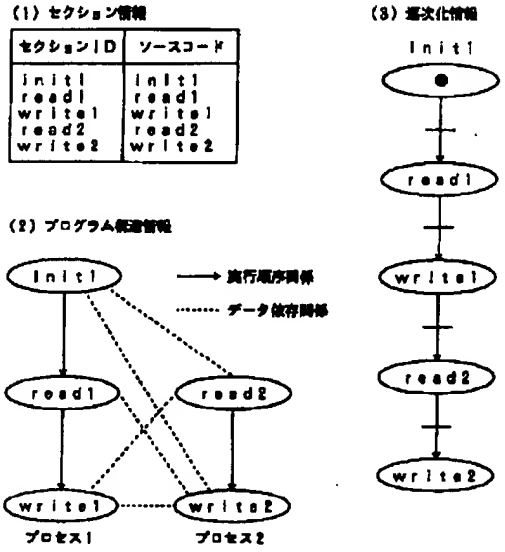
【図10】



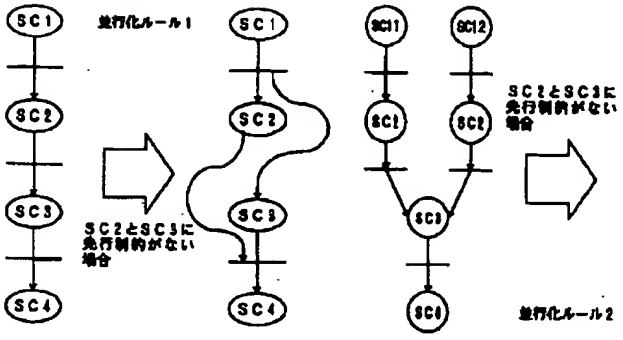
【図11】



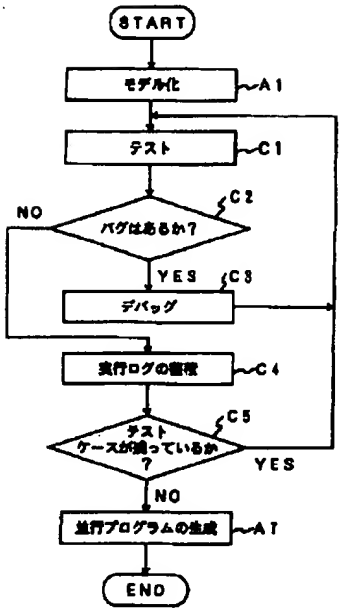
【図15】



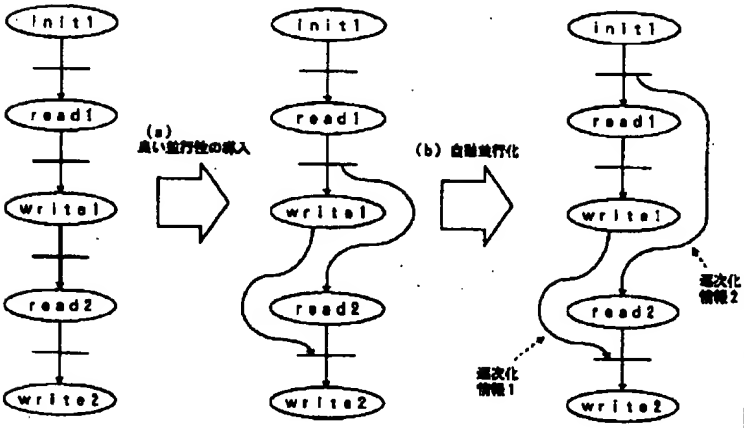
【図１２】



【図２２】



【図１６】

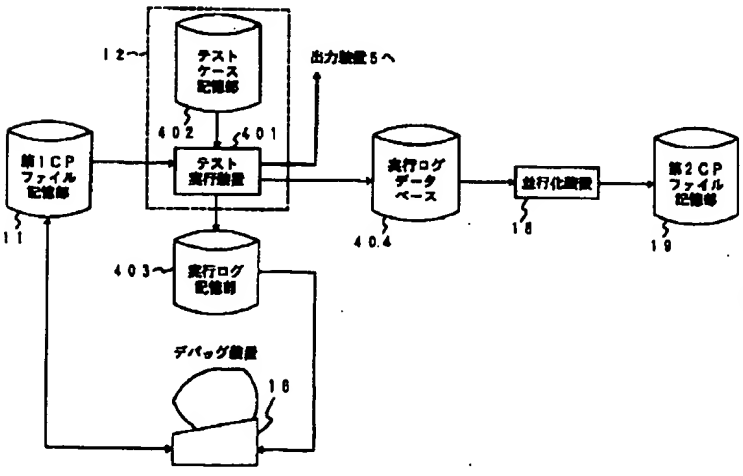


【図２３】

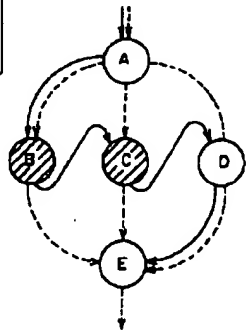
```
CP=P1|P2
P1:
begin
job11;/*仕事１１を実行する*/
job12;/*仕事１２を実行する*/
end

P2:
begin
job21;/*仕事２１を実行する*/
job22;/*仕事２２を実行する*/
end
```

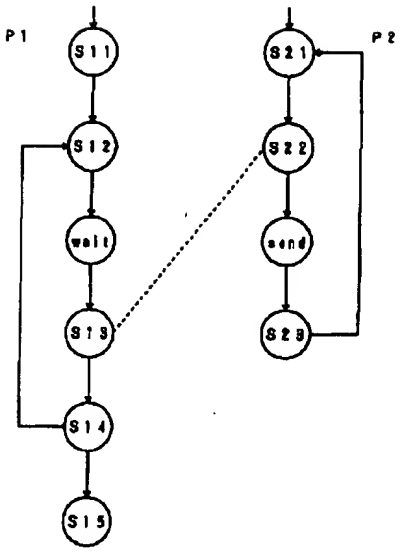
【図２１】



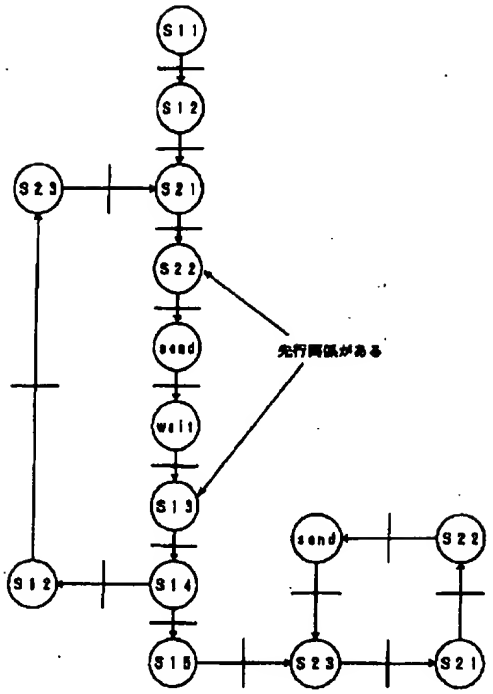
【図３６】



【図18】

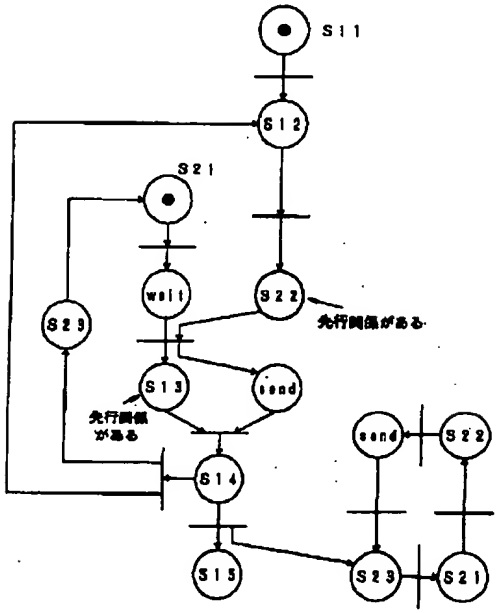


【図19】

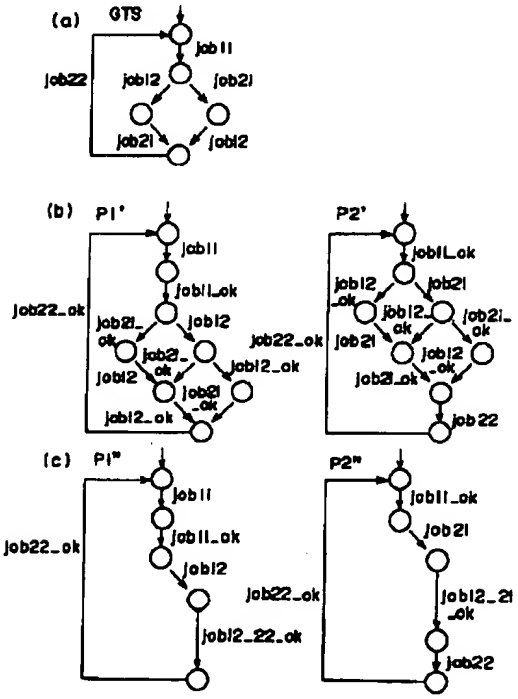


【図42】

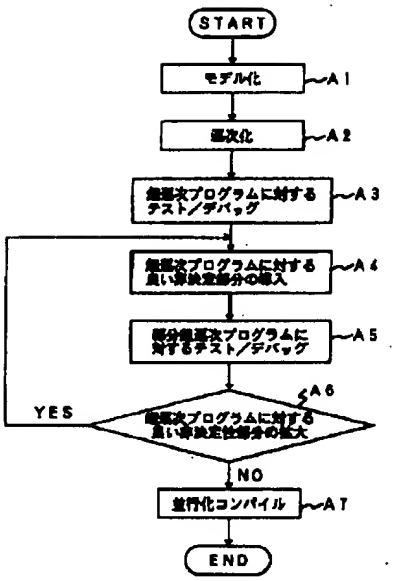
【図20】



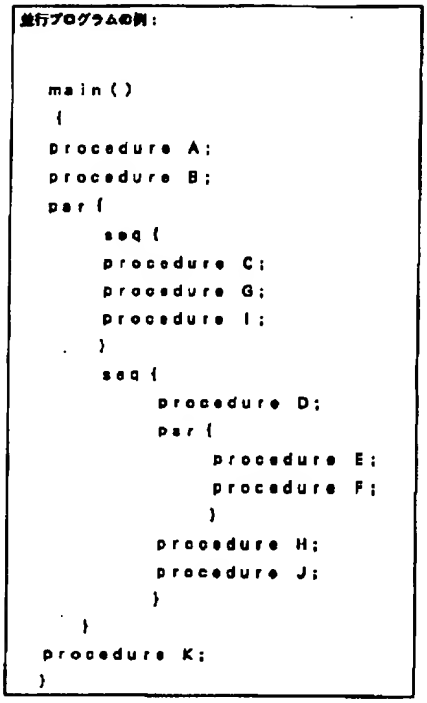
【図24】



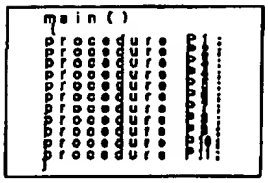
【図25】



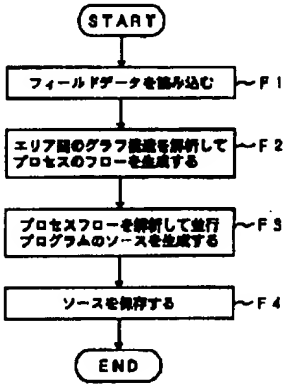
【図27】



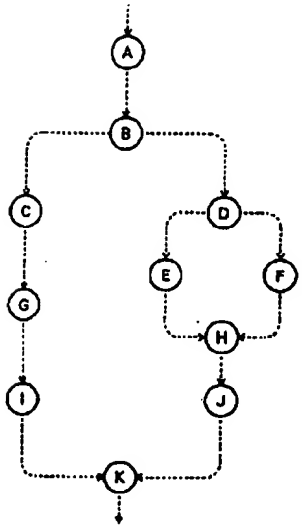
【図41】



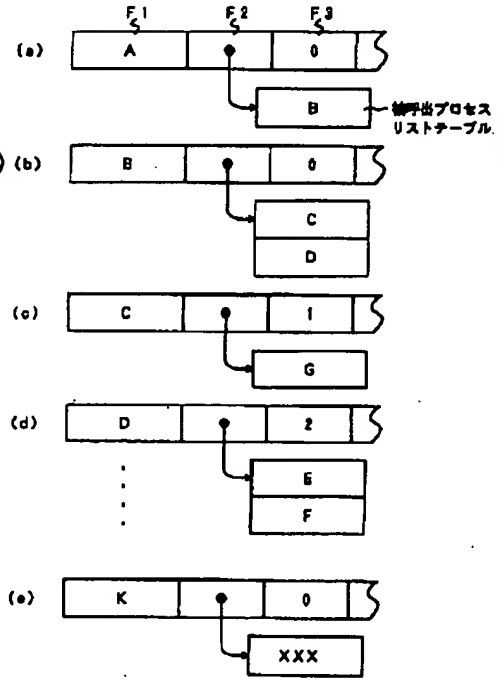
【図67】



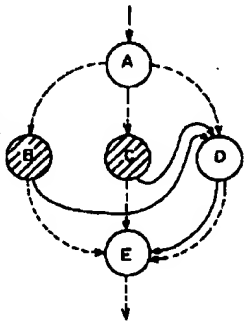
【図28】



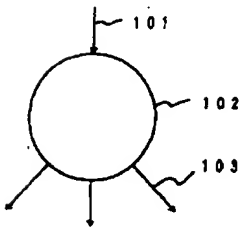
【図29】



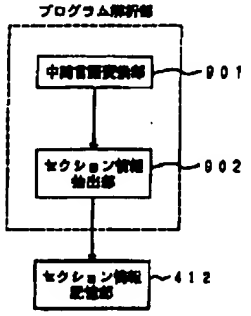
【図37】



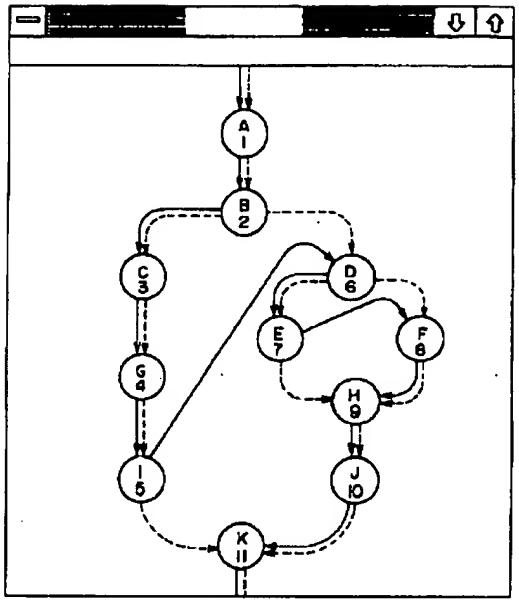
【図100】



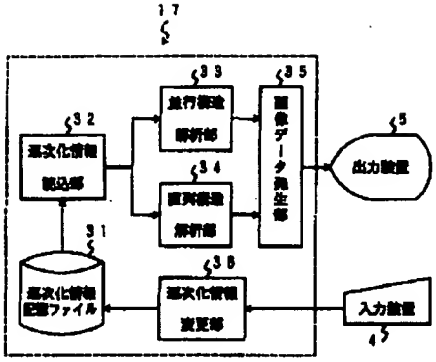
【図108】



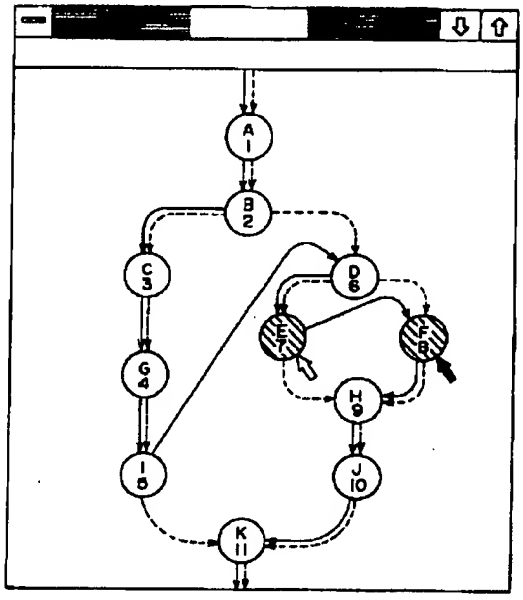
【図３０】



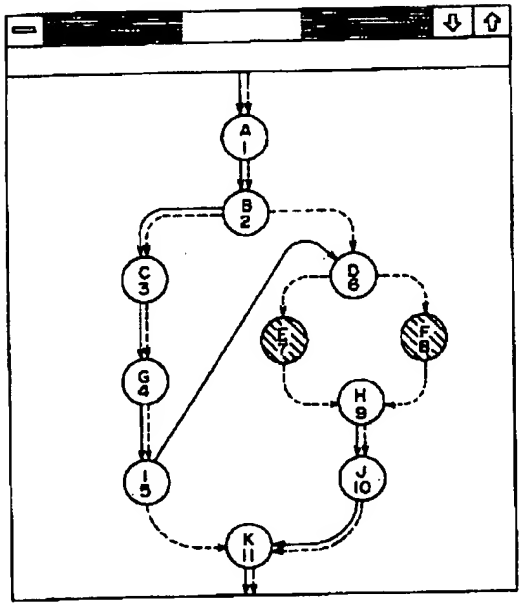
【図３１】



【図３２】



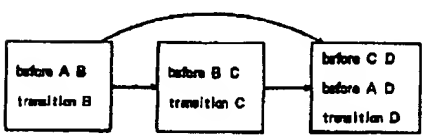
【図３３】



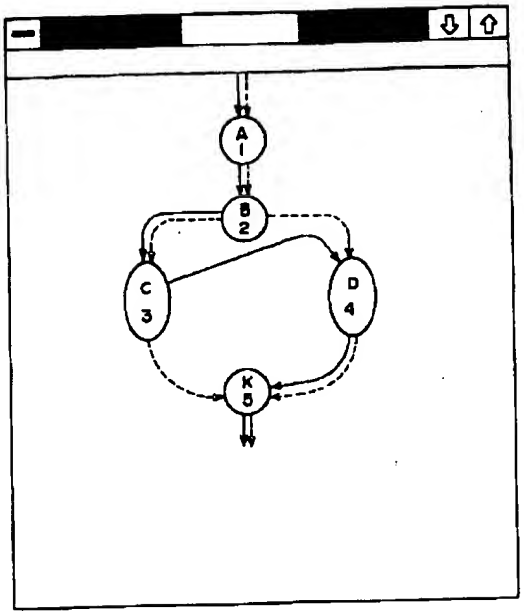
【図３９】

```
main()
procedure P1;
seq(
    procedure P8;
    par(
        procedure P1;
    )
seq(
    procedure P8;
    par(
        procedure P1;
        procedure P8;
    )
procedure P11
```

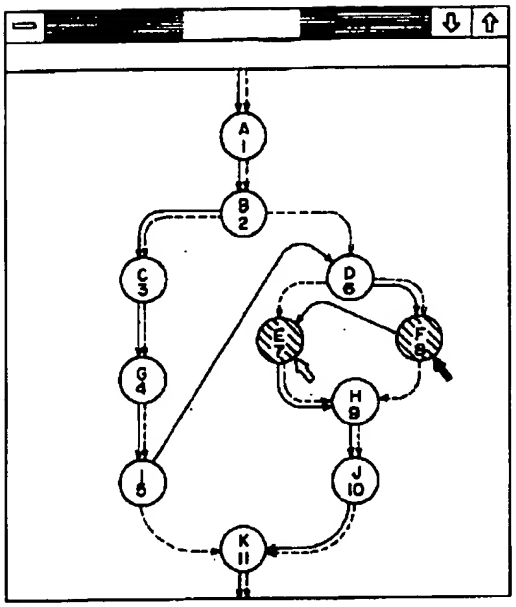
【図５４】



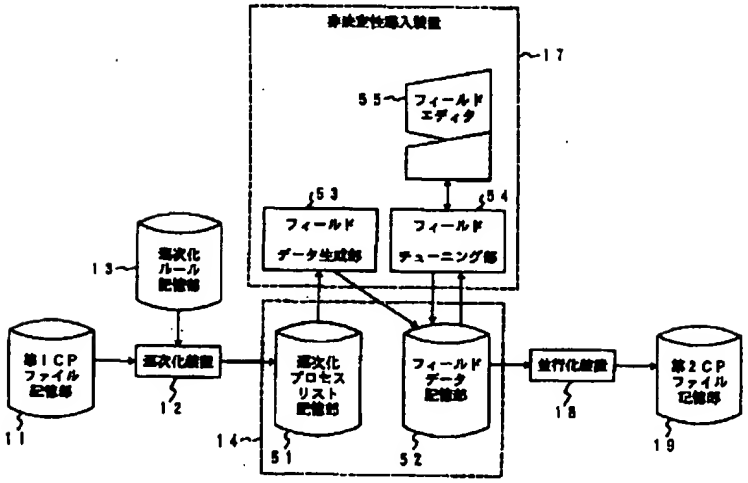
【図３４】


A 1
B 2
C 3
D 4
K 5

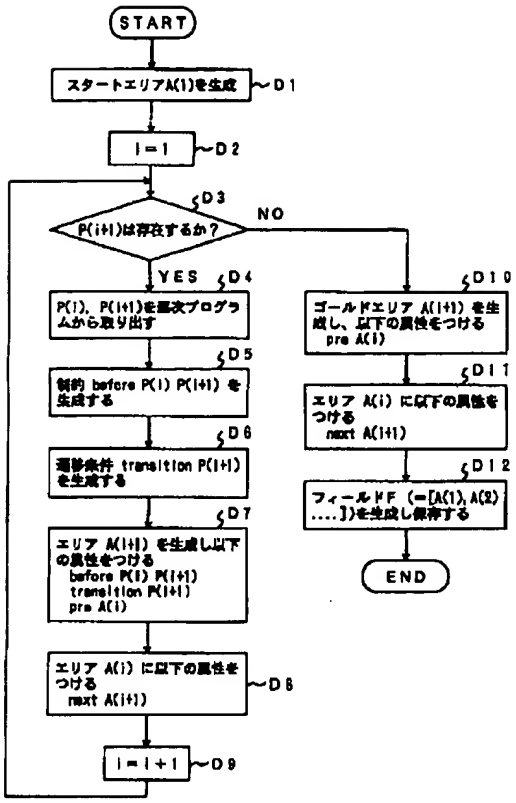
【図３５】


A 1
B 2
C 3
D 6
E 7
F 8
G 4
H 9
I 8
J 10
K 11

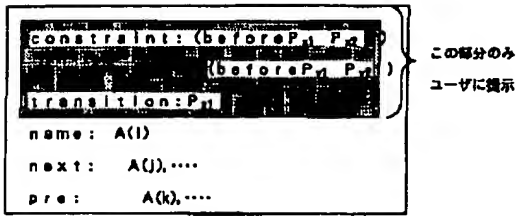
【図３８】


非決定性導入装置
17
55
フィールドエディタ
53
フィールドデータ生成部
54
フィールドチューニング部
13
高次化ルール記憶部
第１CPファイル記憶部
11
高次化機能
12
14
高次化プロセスリスト記憶部
51
フィールドデータ記憶部
52
並行化機能
18
第２CPファイル記憶部
19

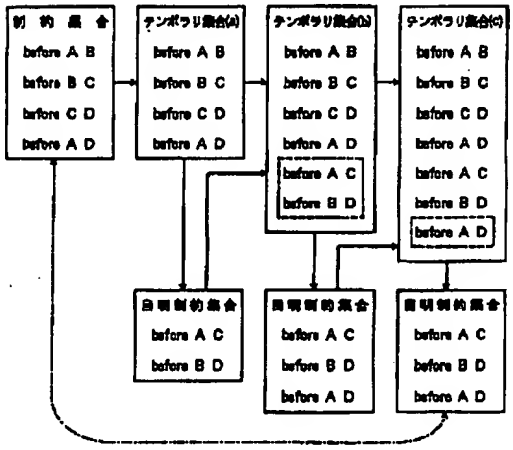
【図４３】



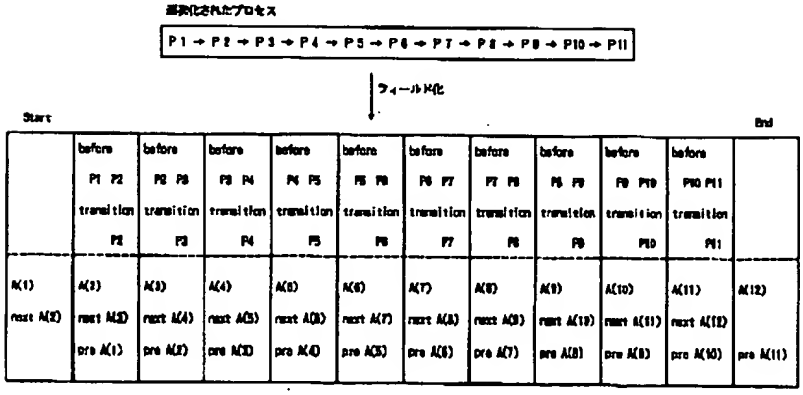
【図４４】



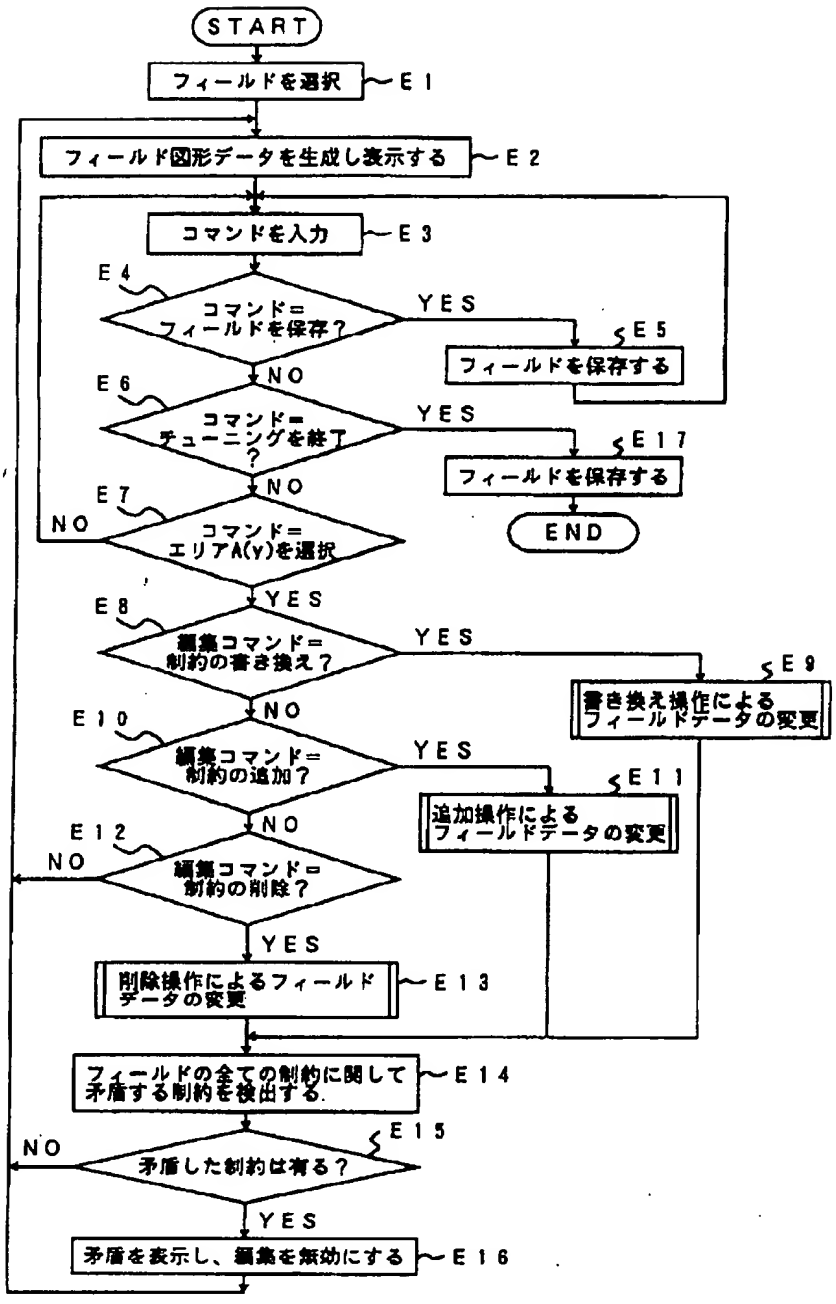
【図５５】



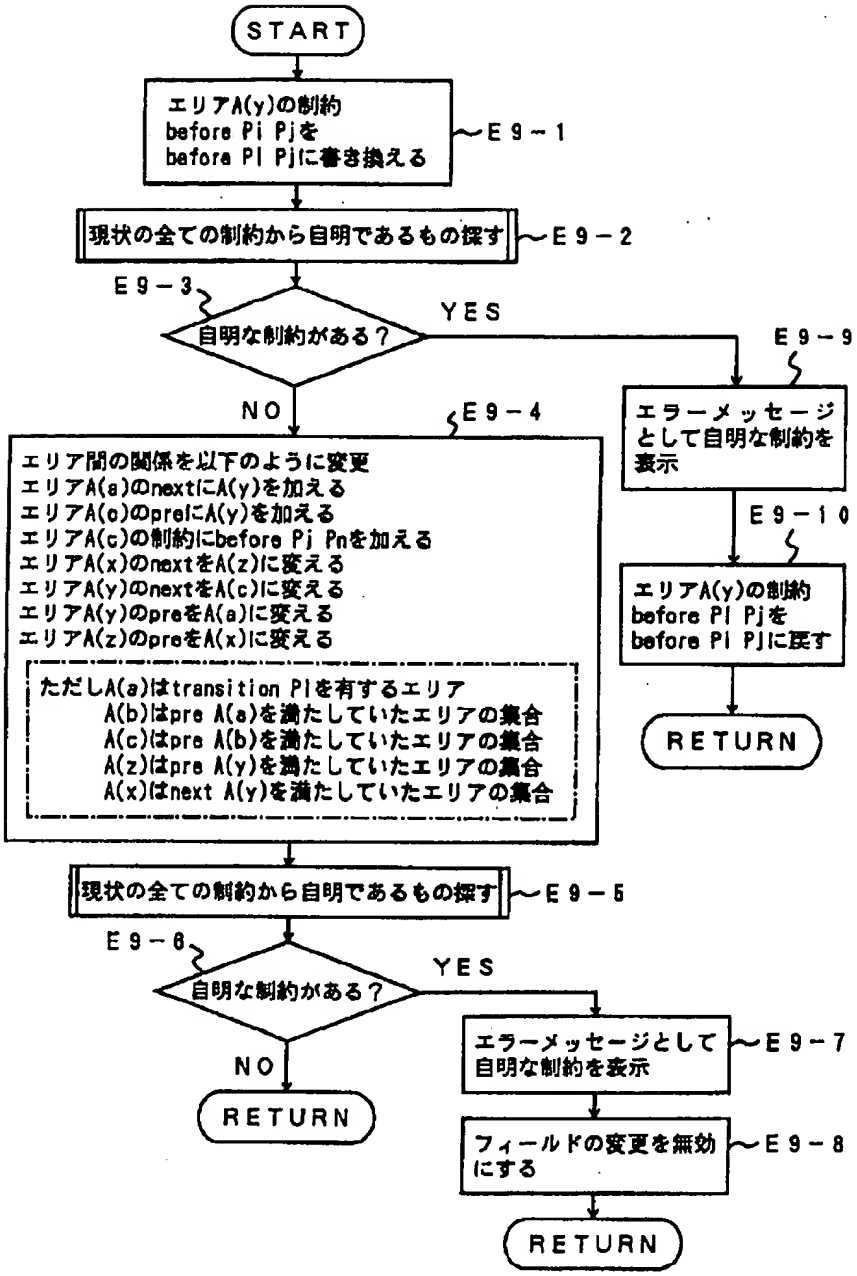
【図４５】



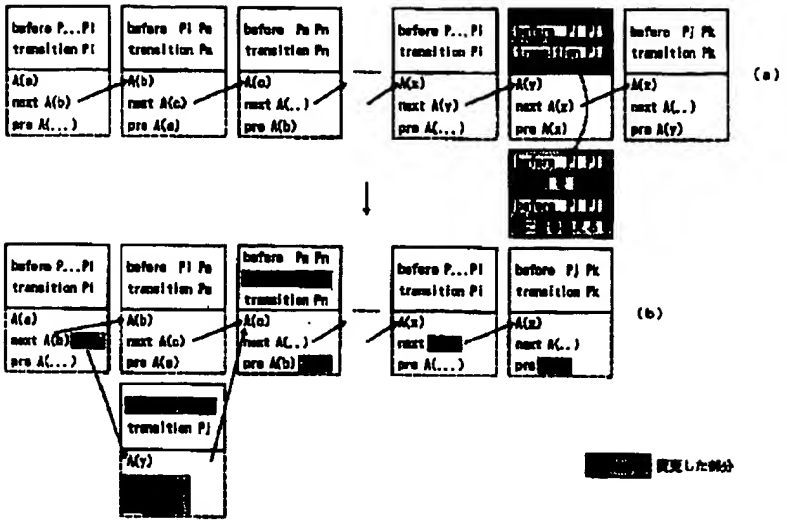
【図46】


START
フィールドを選択
E1
フィールド図形データを生成し表示する
E2
コマンドを入力
E3
E4
コマンド＝フィールドを保存？
YES
E5
フィールドを保存する
NO
E6
コマンド＝チューニングを終了？
YES
E17
フィールドを保存する
END
NO
E7
コマンド＝エリアA(y)を選択
YES
E8
編集コマンド＝制約の書き換え？
YES
E9
書き換え操作によるフィールドデータの変更
NO
E10
編集コマンド＝制約の追加？
YES
E11
追加操作によるフィールドデータの変更
NO
E12
NO
編集コマンド＝制約の削除？
YES
削除操作によるフィールドデータの変更
E13
フィールドの全ての制約に関して矛盾する制約を検出する.
E14
E15
NO
矛盾した制約は有る？
YES
矛盾を表示し、編集を無効にする
E16

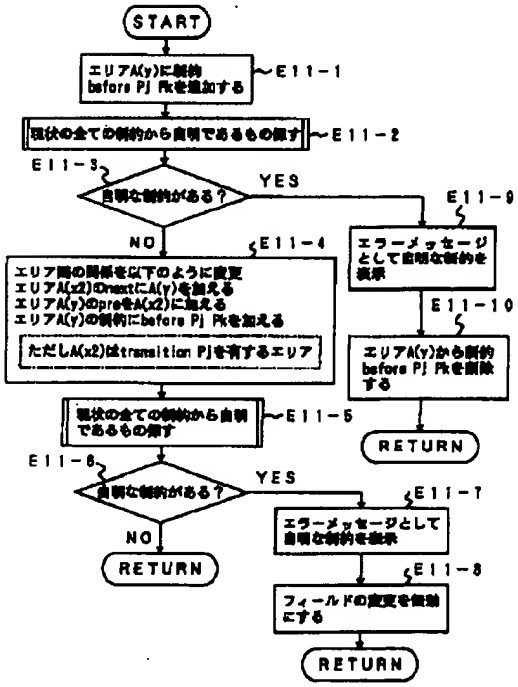
【図47】


START
エリアA(y)の制約
before Pi Pjを
before Pl Pjに書き換える
E9－1
現状の全ての制約から自明であるもの探す
E9－2
E9－3
自明な制約がある？
YES
E9－9
NO
E9－4
エラーメッセージ
として自明な制約を
表示
エリア間の関係を以下のように変更
エリアA(a)のnextにA(y)を加える
エリアA(c)のpreにA(y)を加える
エリアA(c)の制約にbefore Pj Pnを加える
エリアA(x)のnextをA(z)に変える
エリアA(y)のnextをA(c)に変える
エリアA(y)のpreをA(a)に変える
エリアA(z)のpreをA(x)に変える
ただしA(a)はtransition Plを有するエリア
A(b)はpre A(a)を満たしていたエリアの集合
A(c)はpre A(b)を満たしていたエリアの集合
A(z)はpre A(y)を満たしていたエリアの集合
A(x)はnext A(y)を満たしていたエリアの集合
E9－10
エリアA(y)の制約
before Pl Pjを
before Pl Pjに戻す
RETURN
現状の全ての制約から自明であるもの探す
E9－5
E9－6
自明な制約がある？
YES
エラーメッセージとして
自明な制約を表示
E9－7
NO
RETURN
フィールドの変更を無効
にする
E9－8
RETURN

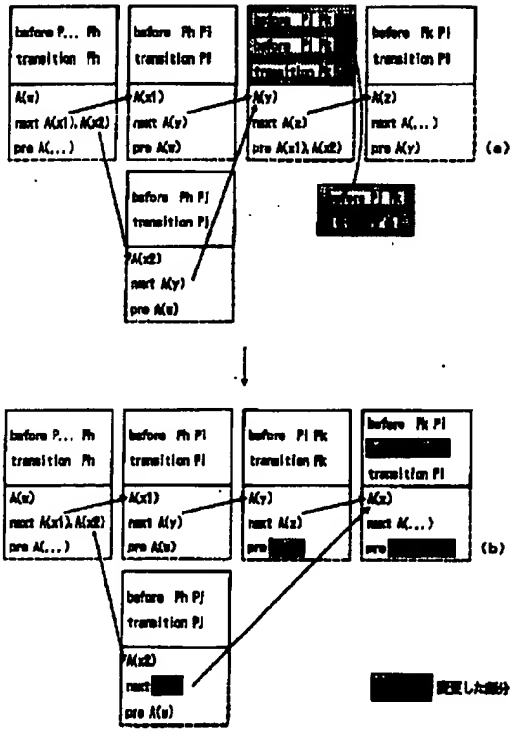
【図48】



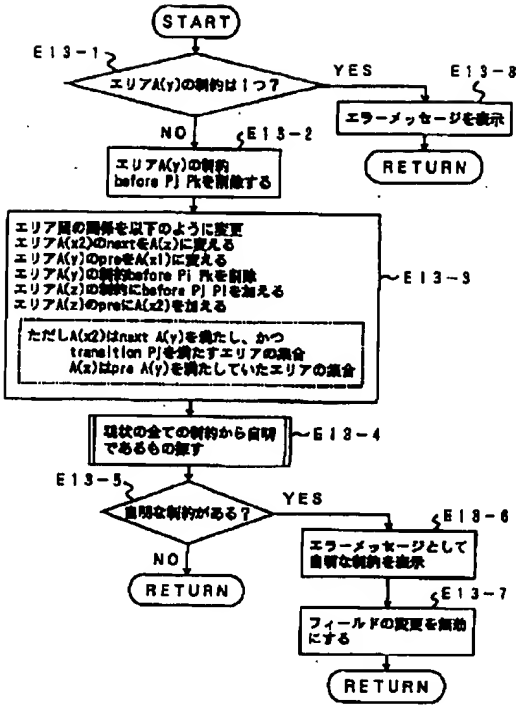
【図49】


START
エリアA(y)に制約 before PJ Pkを追加する
E11-1
現状の全ての制約から自明であるものを除す
E11-2
E11-3
自明な制約がある？
YES
NO
E11-9
エラーメッセージとして自明な制約を表示
E11-4
エリア間の関係を以下のように変更
エリアA(x2)のnextにA(y)を加える
エリアA(y)のpreをA(x2)に加える
エリアA(y)の制約にbefore PJ Pkを加える
ただしA(x2)はtransition PJを有するエリア
E11-10
エリアA(y)から制約 before PJ Pkを削除する
現状の全ての制約から自明であるものを除す
E11-5
RETURN
E11-6
自明な制約がある？
YES
NO
E11-7
エラーメッセージとして自明な制約を表示
RETURN
E11-8
フィールドの変更を無効にする
RETURN

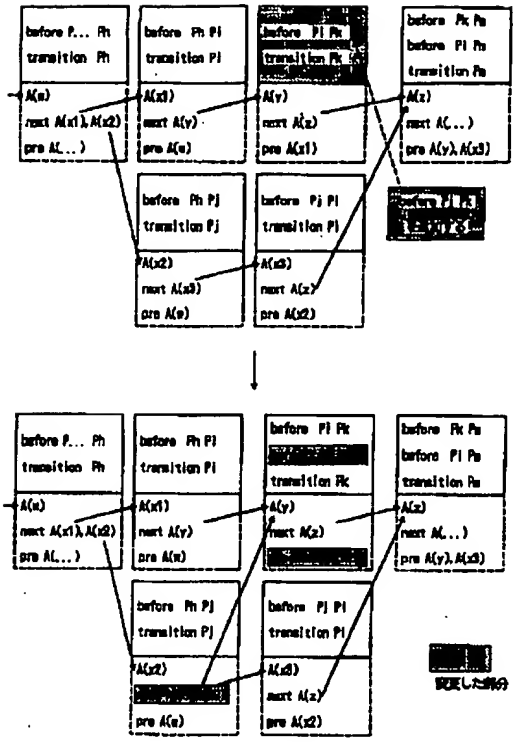
【図50】


(a)
(b)

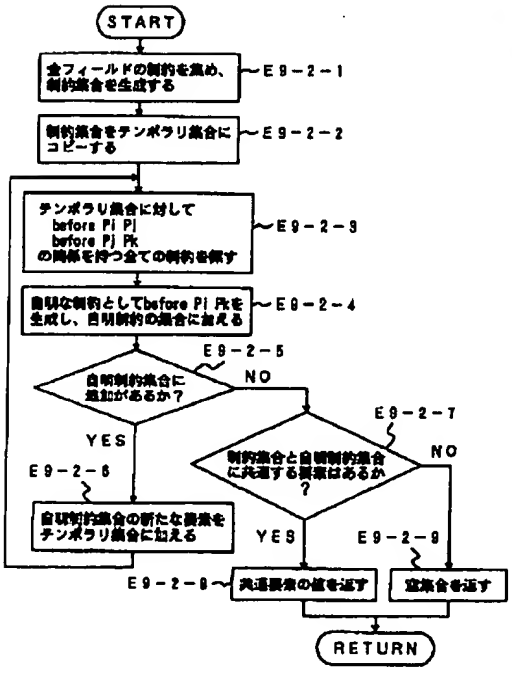
【図５１】



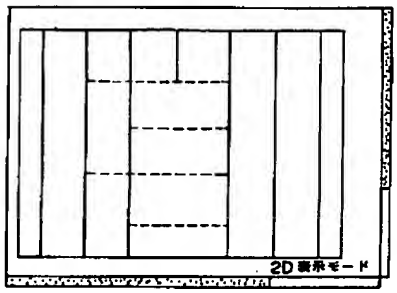
【図５２】



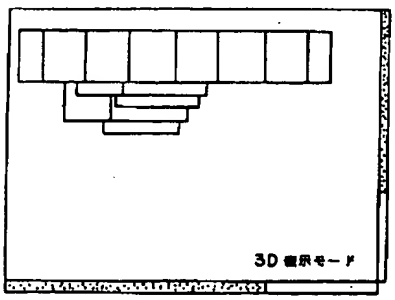
【図５３】



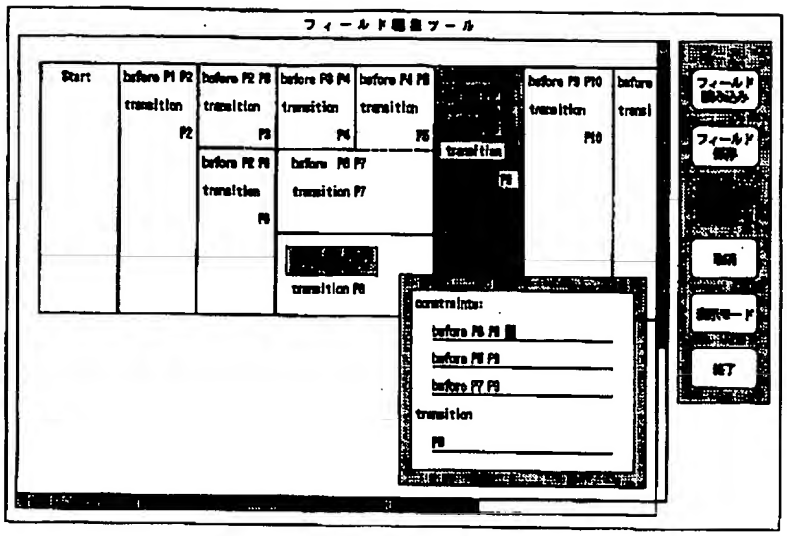
【図５７】



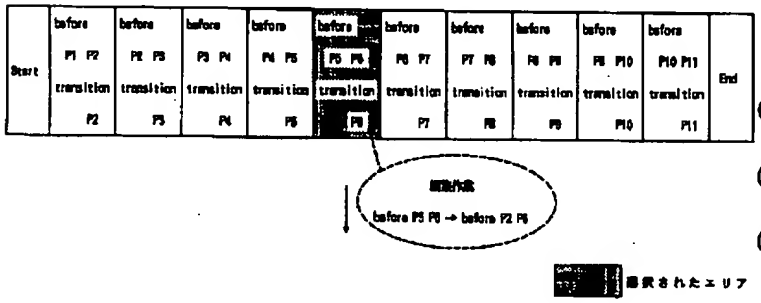
【図５８】



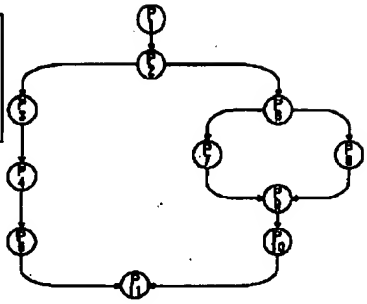
【図５６】



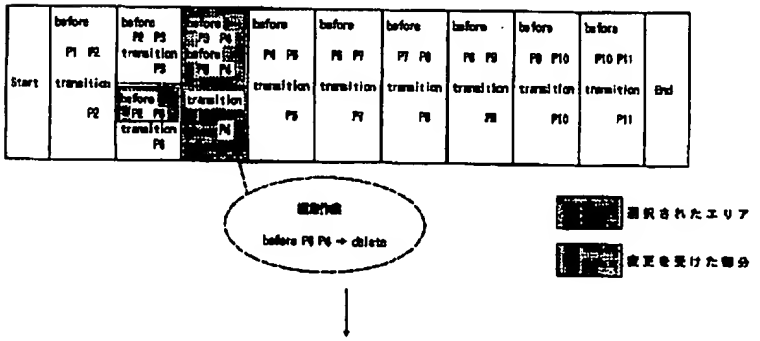
【図５９】



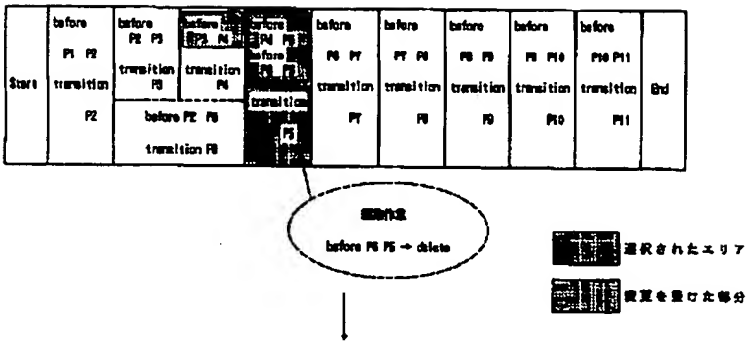
【図６９】



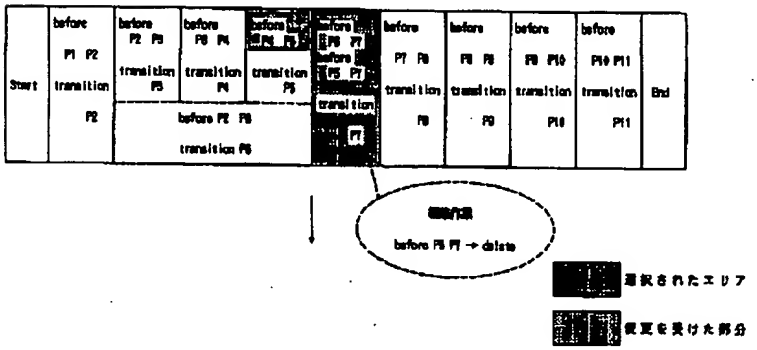
【図６０】



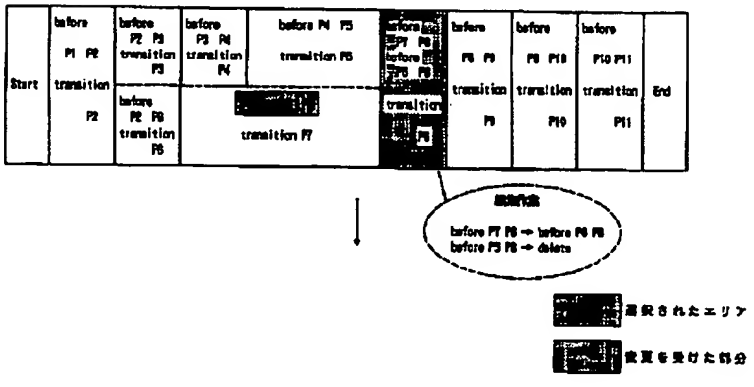
【図６１】



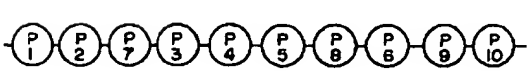
【図６２】



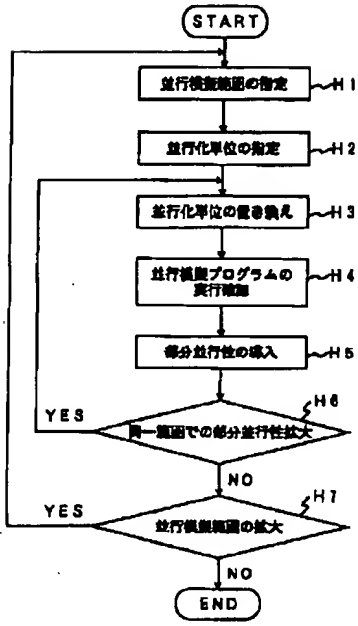
【図６３】



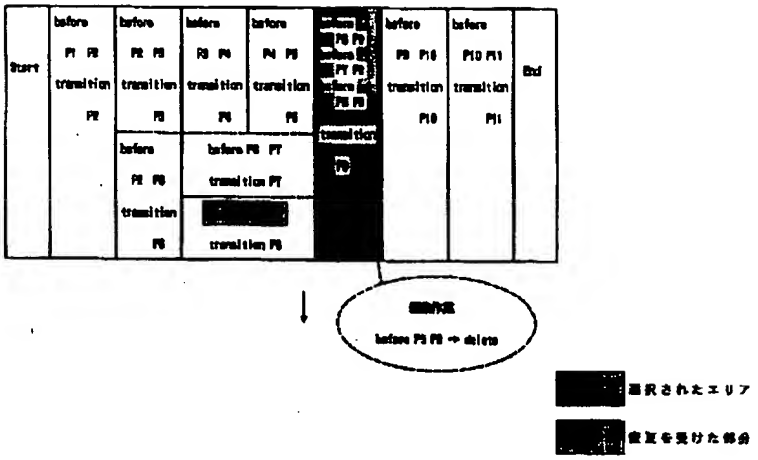
【図７３】



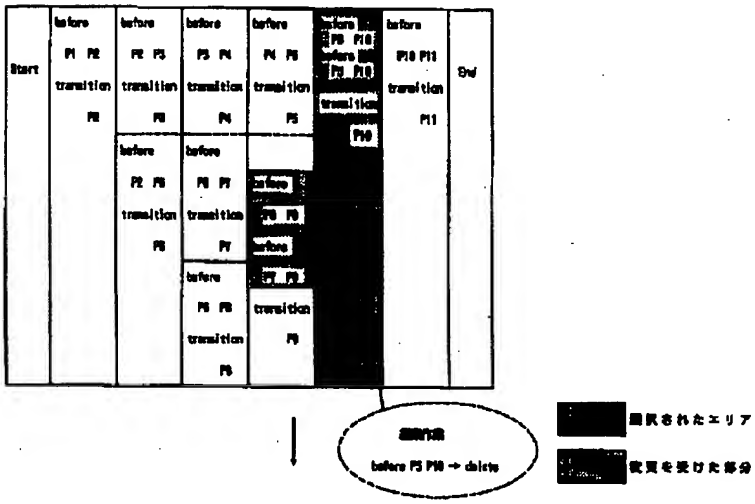
【図７４】



【図64】



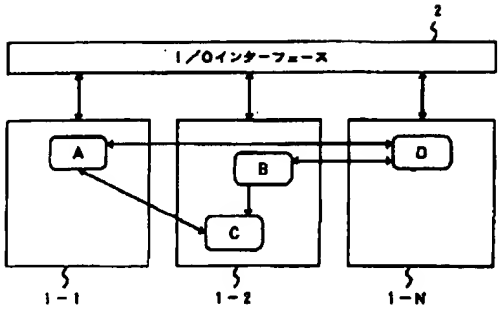
【図65】



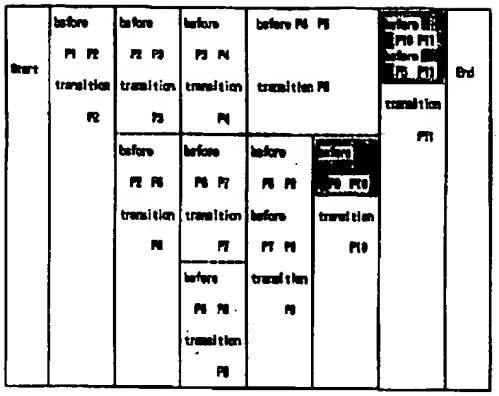
【図68】

```
main()
procedure P1;
procedure P2;
par
        seq
          procedure P3;
          procedure P4;
          procedure P5;
        seq
          procedure P6;
          par
            procedure P7;
            procedure P8;
          procedure P9;
          procedure P10;
procedure P11;
```

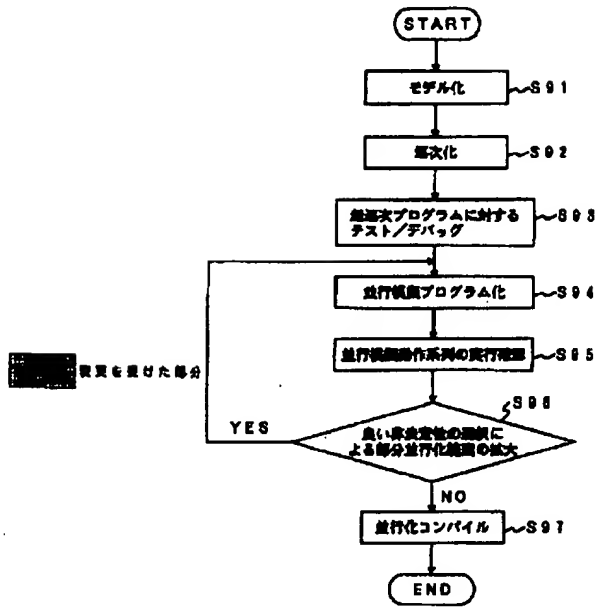
【図79】



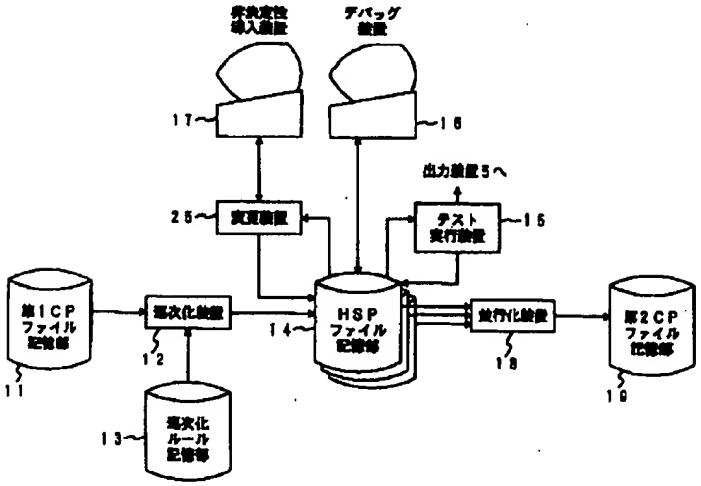
【図６６】



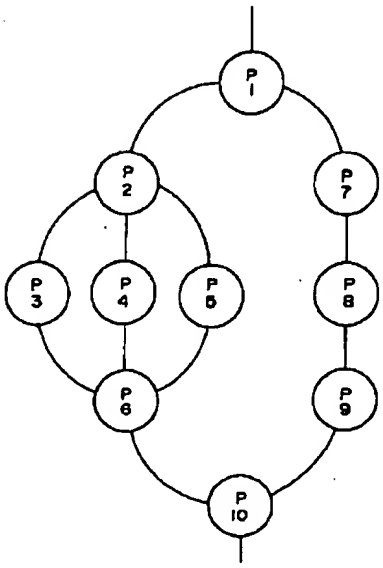
【図７１】



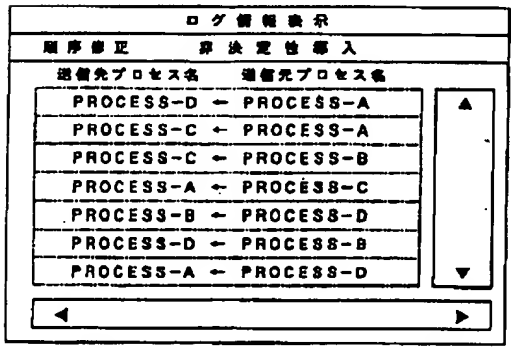
【図７０】



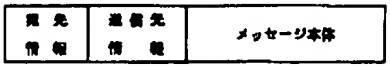
【図７２】



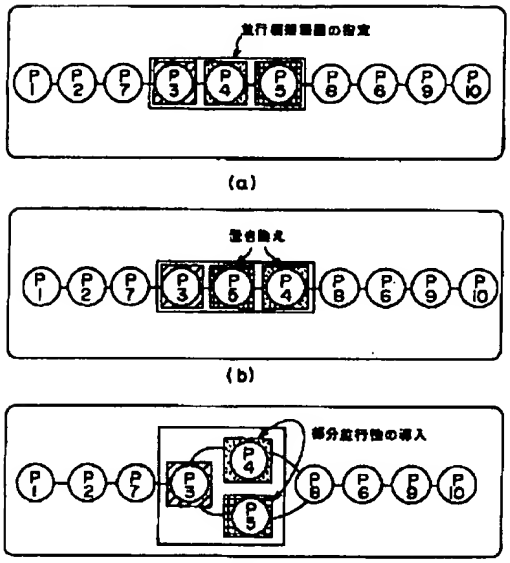
【図８２】



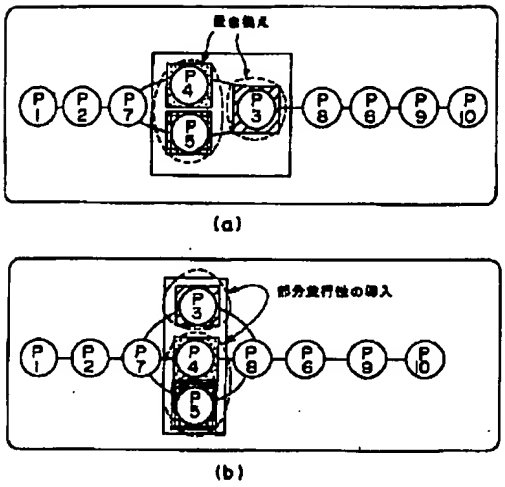
【図８４】



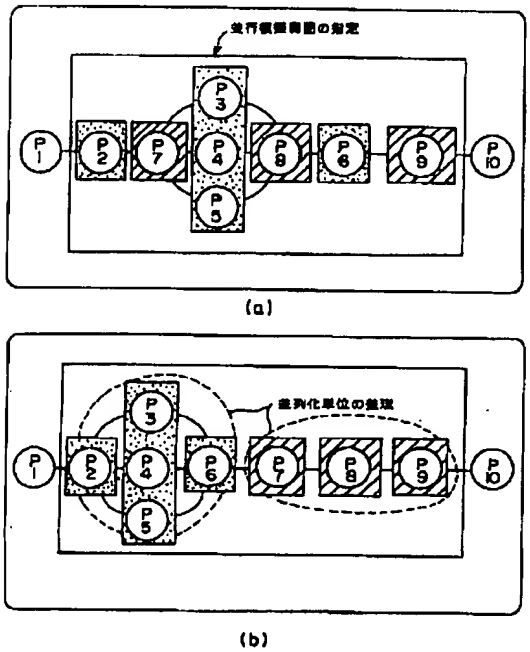
【図75】



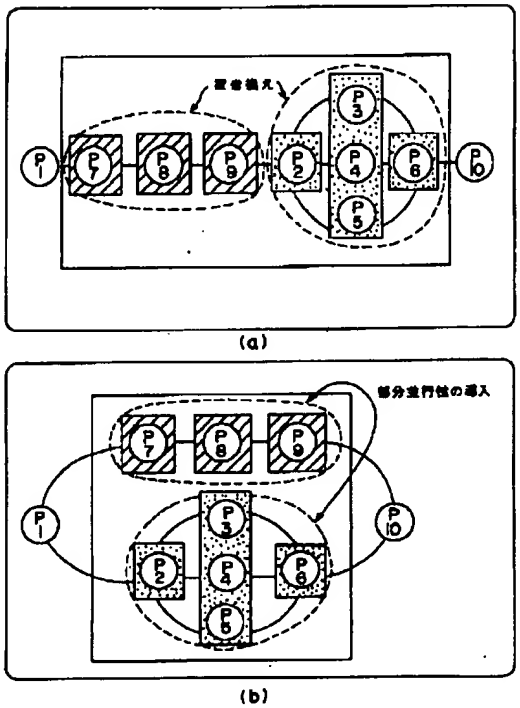
【図76】



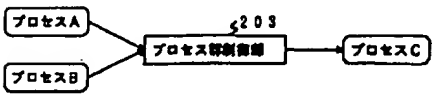
【図77】



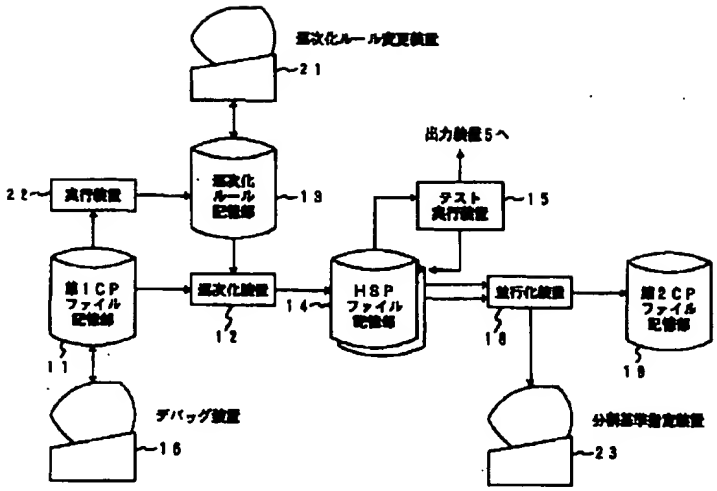
【図78】



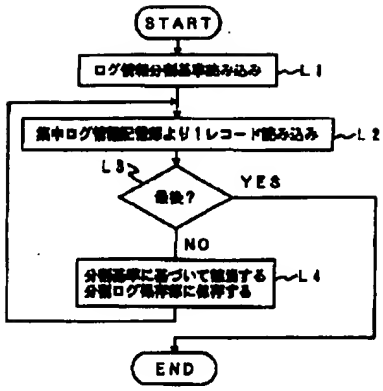
【図85】



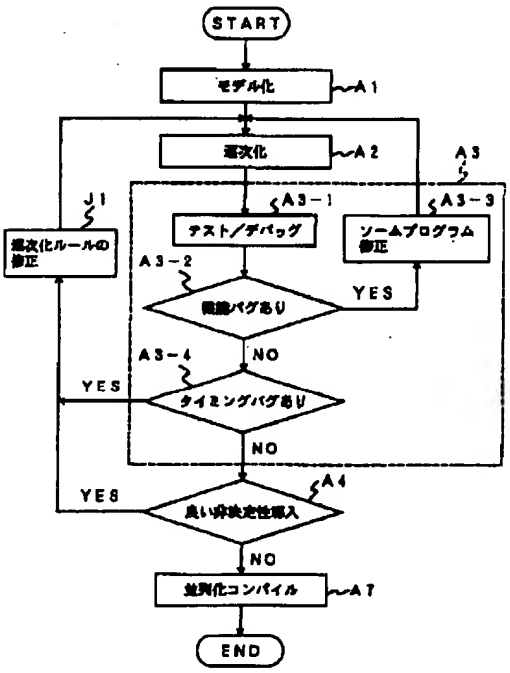
【図80】



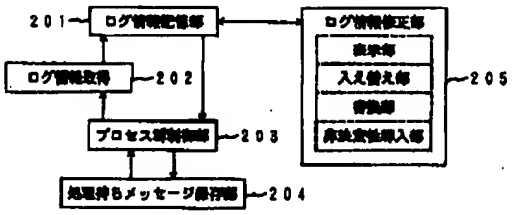
【図95】



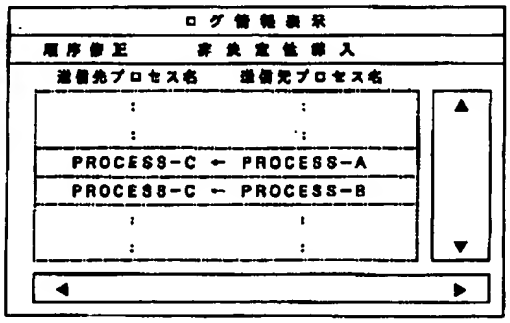
【図81】



【図83】



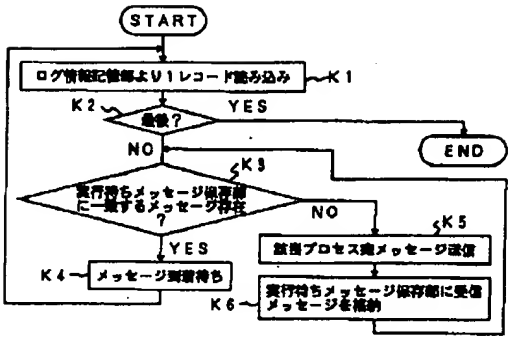
【図86】



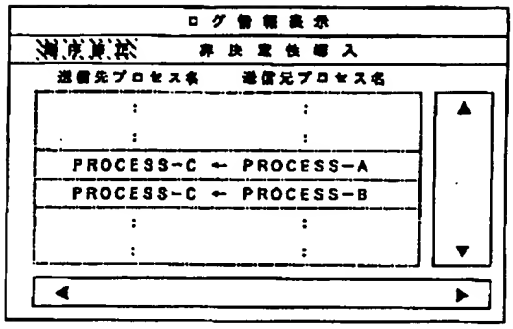
【図97】

```
プロセス処理( )
{
    call (プロセス制御手続) ;
        :
      プロセス本体処理
        :
}
```

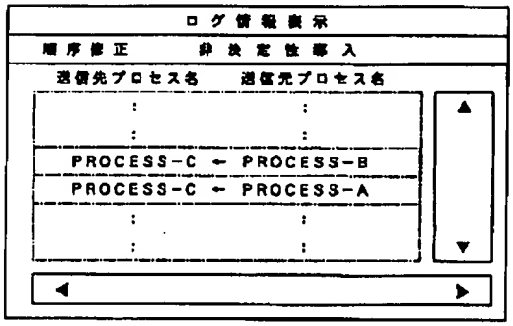
【図87】



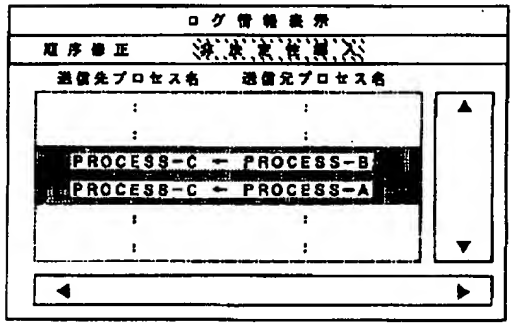
【図88】



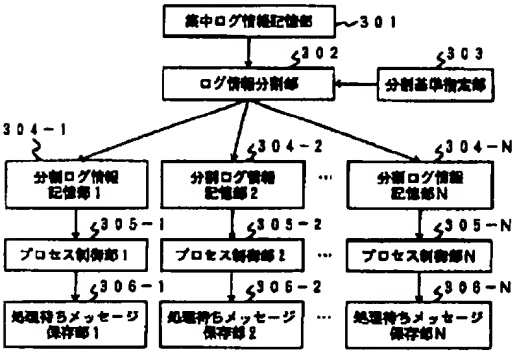
【図89】



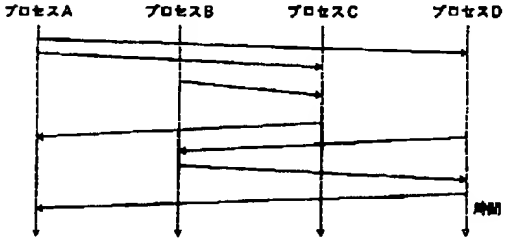
【図90】



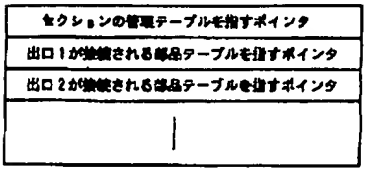
【図92】



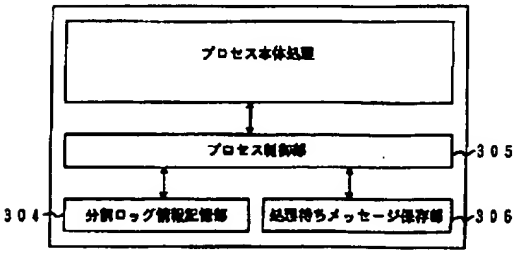
【図93】



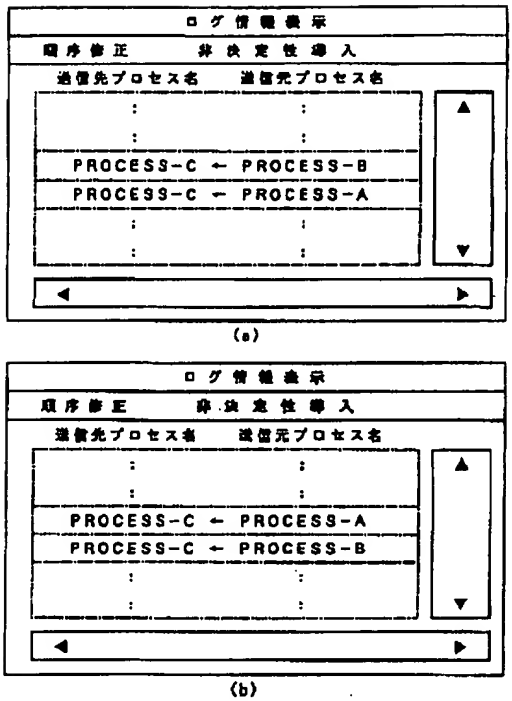
【図106】



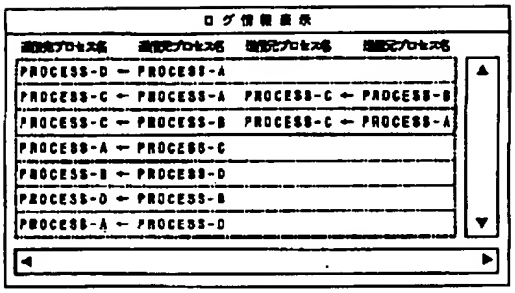
【図99】



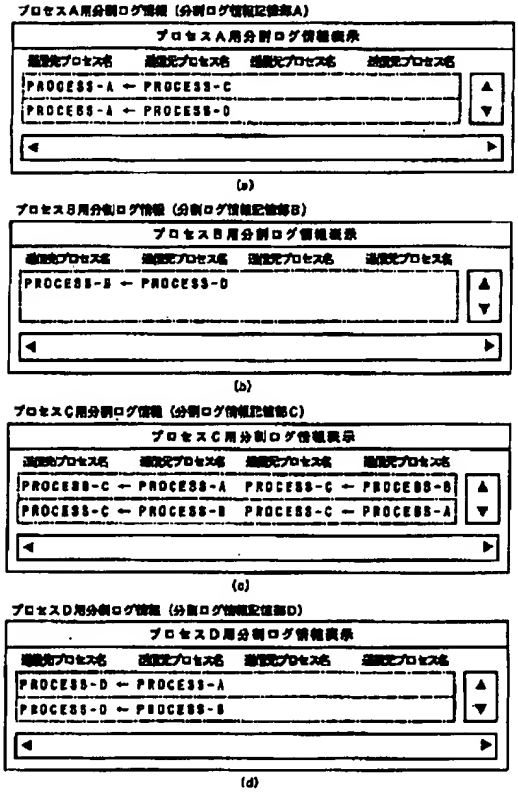
【図９１】



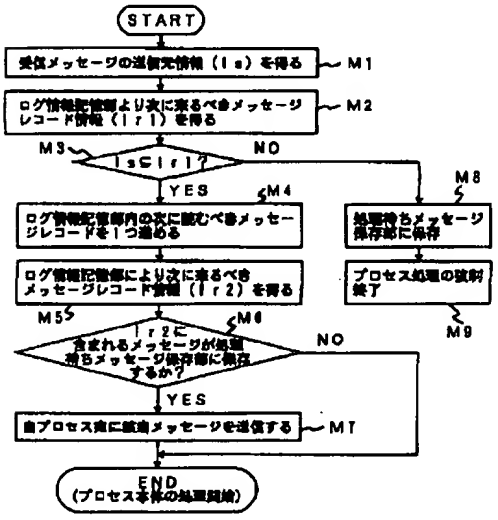
【図９４】



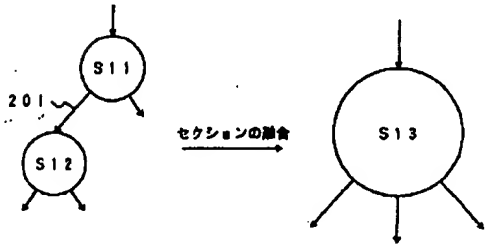
【図９６】



【図９８】

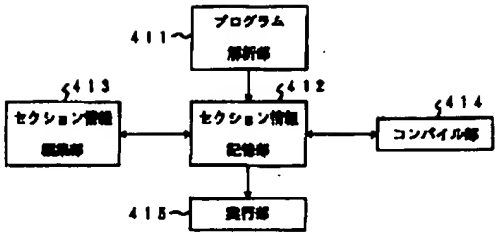


【図１０１】

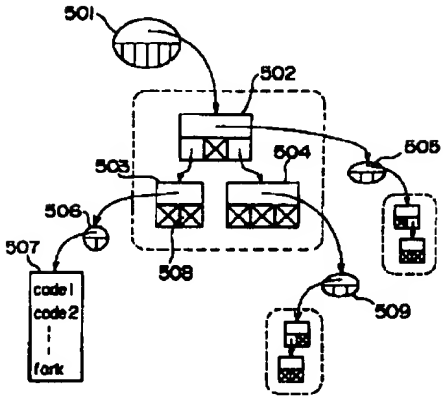


【図１０２】

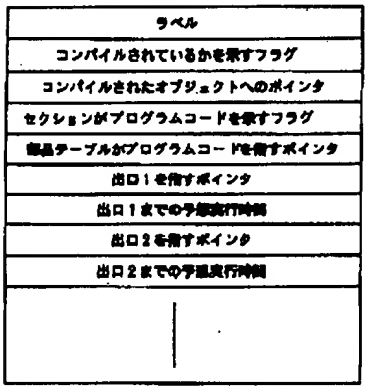
【図１０３】



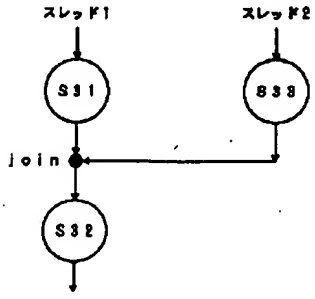
【図１０４】



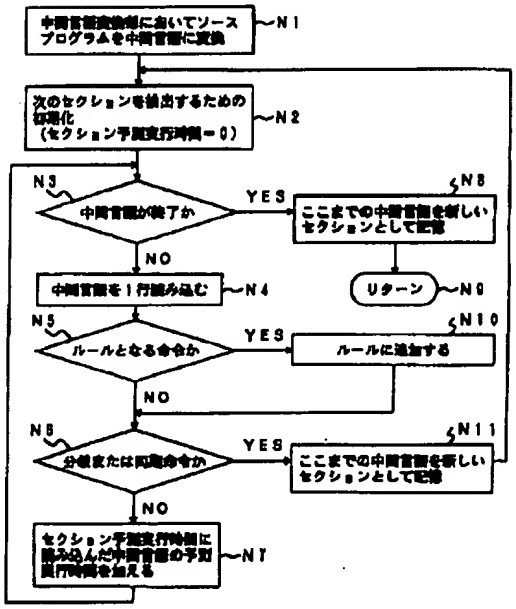
【図１０５】



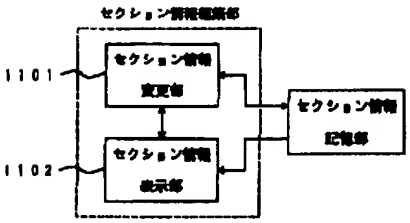
【図１０７】



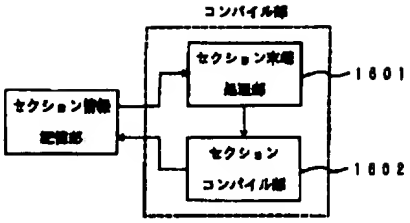
【図１０９】



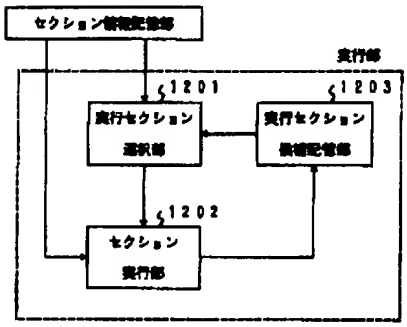
【図１１０】



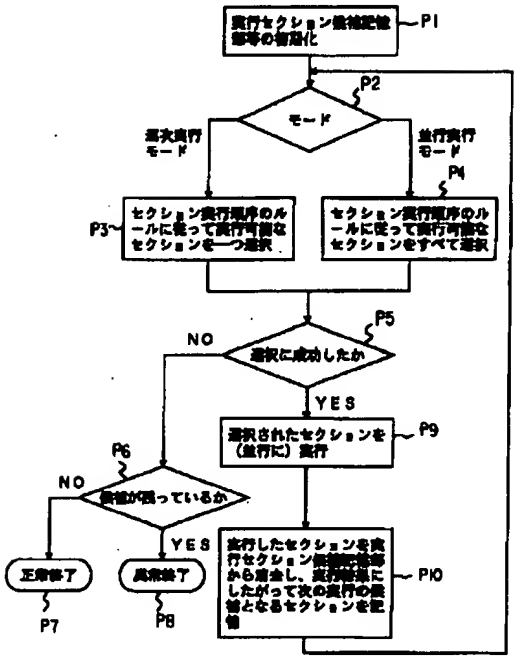
【図１１１】



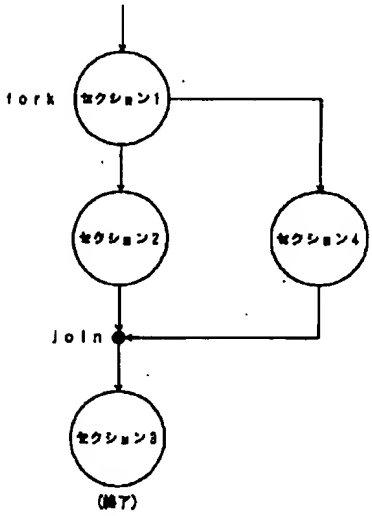
【図１１２】



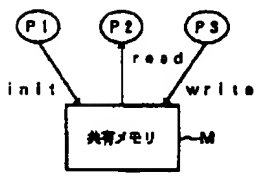
【図１１３】



【図１１４】



【図１１６】

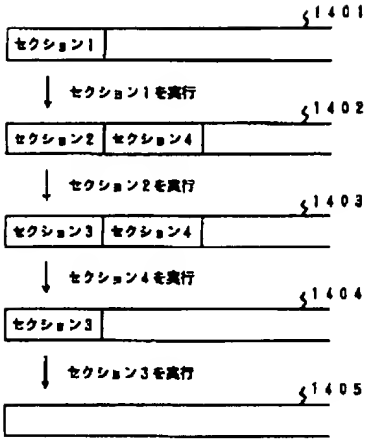


(a)

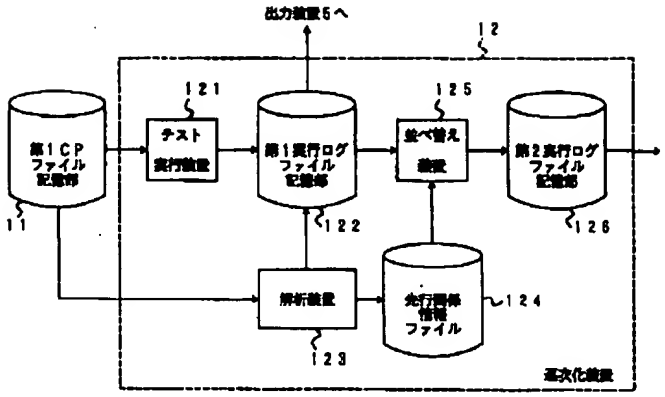
(b)

init → read → write
init → write→ read
read → init → write
read → write→ init
write→ init → read
write→ read → init

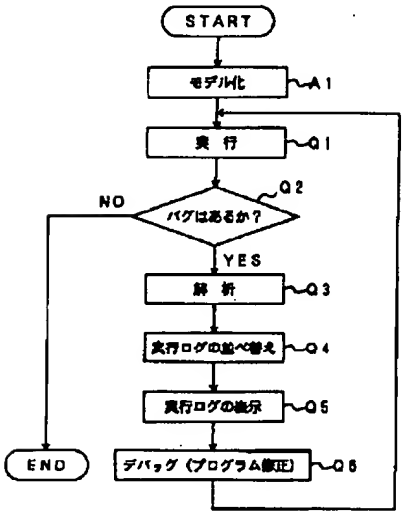
【図１１５】



【図１１７】



【図１１８】



【図１１９】

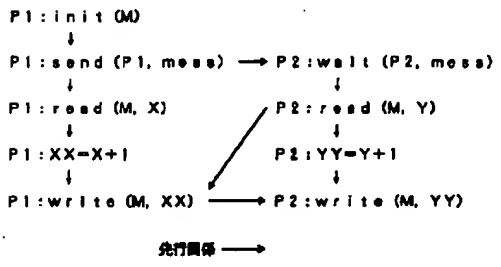
```
CP=P1|P2
P1:
begin
extern int M;
int X, XX;
init (M) ;
send (P2, mess) ;
read (M, X) ;
XX=X+1;
end

P2:
begin
extern int M;
int Y, YY;
wait (P1, mess) ;
read (M, Y) ;
YY=Y-1;
write (M, YY) ;
end
```

【図１２０】

```
実行ログ:
P1:init (M) ;
P1:send (P2, mess) ;
P1:read (M, X) ;
P2:wait (P1, mess) ;
P1:XX=X+1;
P2:read (M, Y) ;
P2:YY=Y-1;
P1:write (M, XX) ;
P2:write (M, YY) ;
```

【図１２１】



【図１２２】

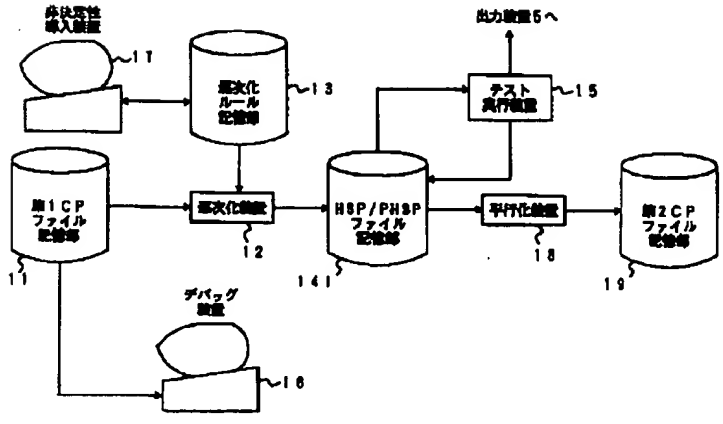
```
並び替えた実行ログ：
P1:init (M) ;
P1:send (P2, mess) ;
P1:read (M, X) ;
P1:XX=X+1;
P2:wait (P1, mess) ;
P2:read (M, Y) ;
P1:write (M, XX) ;
P2:YY=Y-1;
P2:write (M, YY) ;
```

【図１２３】

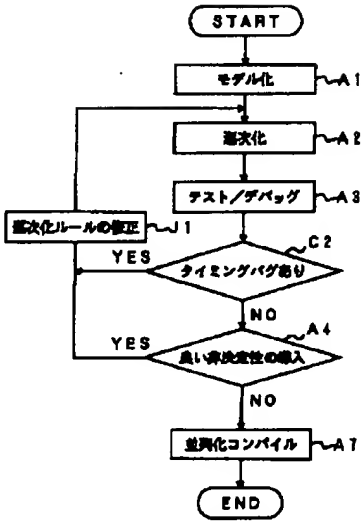
```
CP=P1|P2
P1:
begin
extern int M;
int X, XX;
init (M) ;
read (M, X) ;
XX=X+1;
write (M, XX) ;
send (P2, mess) ;
end

P2:
begin
extern int M;
int Y, YY;
wait (P1, mess) ;
read (M, X) ;
YY=Y-1;
write (M, YY) ;
end
```

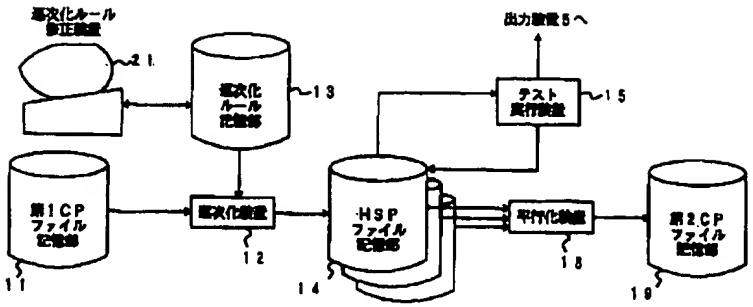
【図１２４】



【図１２５】



【図126】



フロントページの続き

(72)発明者　關　俊文
神奈川県川崎市幸区柳町70番地　株式会社東芝柳町工場内
(72)発明者　永井　保夫
神奈川県川崎市幸区柳町70番地　株式会社東芝柳町工場内
(72)発明者　半田　恵一
神奈川県川崎市幸区柳町70番地　株式会社東芝柳町工場内
(72)発明者　伊藤　聡
神奈川県川崎市幸区柳町70番地　株式会社東芝柳町工場内
(72)発明者　湯島　慎介
神奈川県川崎市幸区小向東芝町１番地　株式会社東芝研究開発センター内
(72)発明者　田原　康之
神奈川県川崎市幸区柳町70番地　株式会社東芝柳町工場内
(72)発明者　塩谷　英明
神奈川県川崎市幸区柳町70番地　株式会社東芝柳町工場内